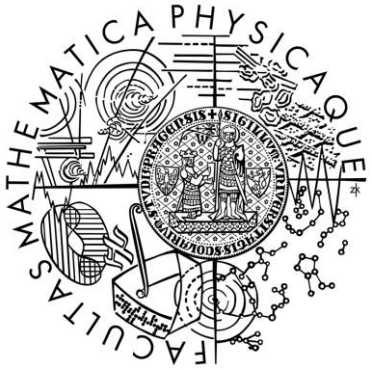
Charles University in Prague

Faculty of Mathematics and Physics

MASTER THESIS



Jakub Čermák

Rozpoznávání drah částic v pixelovém detektoru typu Timepix

Kabinet software a výuky informatiky

Supervisor of the master thesis: Ing. Pavel Čermák, Ph.D.

Study programme: Informatics

Specialization: ISS

Prague 2013

I would like to express my gratitude to my supervisor Ing. Pavel Čermák, PhD and to Ing. Petr Dokládal, PhD for the valuable advice and assistance they have provided and to Y. Shitov, who provided me simulated data. Also, I would like to thank the Institude of Experimental and Applied Physics of the Czech Technical University in Prague for the opportunity to participate in a physical research.

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular the fact that the Charles University in Prague has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 paragraph 1 of the Copyright Act.

In…..... date..... Jakub Čermák

Název práce: Rozpoznávání drah částic v pixelovém detektoru typu Timepix
Autor: Jakub Čermák
Katedra / Ústav: Kabinet software a výuky informatiky
Vedoucí diplomové práce: Ing. Pavel Čermák, Ph.D., Ústav technické a experimentální fyziky, České vysoké učení technické v Praze

Abstrakt:
V současné fyzice částic se používají progresivní detekční technologie, k nimž se řadí i pixelové detektory. Tyto detektory jsou rozděleny na malé subdetektory (tzv. pixely), což umožňuje sledovat přesně dráhy detekovaných částic. Tato práce definuje kritéria pro matematický popis tvaru drah částic různých typů (e^-, γ, p, α, μ) a testuje a srovnává vybrané metody používané pro klasifikaci – neuronové sítě, rozhodovací stromy a další. V rámci práce byl vytvořen software Pixa pro komplexní zpracování dat z pixelových detektorů, který implementuje tyto charakteristiky a klasifikační metody a umožňuje vytvářet statistiky a další fyzikální výstupy.

Klíčová slova: rozpoznávání tvarů, pixelový detektor, klasifikační metody

Title: Particle track recognition in Timepix pixel detector
Author: Jakub Čermák
Department / Institute: Department of Software and Computer Science Education
Supervisor of the master thesis: Ing. Pavel Čermák, Ph.D., Institude of experimental and applied physics, Czech Technical University in Prague

Abstract:
In current particle physics field, the progressive detection technologies are used. The pixel detectors are one of them. These detectors are divided into small subdetectors (pixels), which allow viewing exact tracks of the detected particles. This thesis defines criteria for mathematical description of the shape of the particle tracks of different kinds (e^-, γ, p, α, μ) and compares several methods used for a classification – neural networks, decision trees and others. The Pixa software was implemented to process the data measured by pixel detectors. This software implements the characteristics and classification methods and creates statistical and other physical results.

Keywords: pattern recognition, pixel detector, classification methods

Contents

1 Introduction

Nowadays, semiconductor pixel detectors are important type of sensors used in many fields of natural and technical sciences. In this thesis, the hybrid pixel detector Timepix [1] [2] is used to visualize and study in details signals from different ionizing particles. The Timepix detector is a 2D array of 256*256 single detectors (called pixels) with a pitch of 55 μm. Each pixel provides information on partial energy deposited within the pixel volume by an ionizing particle crossing the detector. This information is collected for some time (so-called exposure time or shutter time). Therefore, we get a 2D image (so-called a frame) of the same size as detector (256*256) containing tracks of ionizing particles collected during the exposure time.

Each particle type behaves differently in the detector material, therefore it creates more or less different track. The shape of the track depends on many parameters like material of detector, kinetic energy of the particle and its ionization capabilities, given mainly by its charge and mass. This thesis uses various mathematical methods to analyze signals from pixel detector. For this purpose, the Pixa application was developed. This software allows to analyze measured data, frame by frame, cluster by cluster, and to visualize physical and morphological properties of the particle track. All those analyses are used to create a model of the particle type and to classify the particle with different kind of data mining and classification methods. This analysis is important for a processing of the experimental data.

The Pixa software was designed to serve as a general toolkit for data measured with a Timepix detector (or any similar device). Therefore, the main architecture requirement for developed software is extensibility. At present, it is used for processing data from the COBRA [3] (CdTe detector) and TGV [4] (Si detector) experiments, so the procedures are adapted to the requirements of those experiments. Both experiments focus on the study of the double-beta decay processes. Both processes have different detectable signals, in COBRA it is two electrons with sum energy ~2.8 MeV, while in TGV it is two X-rays with energy ~21 keV each.

The particle type classification uses data mining methods that are used in many applications; therefore, they are documented and studied very well. They are used in both commercial (e.g. risk analysis for banks, market analysis, and customer

behavior) and scientific (e.g. genetics, medicine, visual or audio data recognition) fields. Generally, the data mining is a process of finding some patterns in large data sets. Pixa implements supervised learning to identify the patterns of tracks of each particle type.

The main goals of the thesis are to evaluate different approaches and parameters to achieve the best results, implement the software and choose the most suitable solution and parameters.

The thesis is divided into 9 chapters. Chapter 2 describes basic facts about pixel detectors, recognition methods and used software technologies. Chapters 3 and 4 are the most important parts of thesis, they include detailed information about software implementation and results of comparison of different classification methods applied on both simulated and experimental data. Chapter 5 contains conclusion, while chapters 6-9 are additional sections.

2 Problem analysis

This chapter describes the problem solved by this thesis and its scientific context. In addition, it includes descriptions of the mathematical and computer methods that were evaluated to achieve required goals.

2.1 Physical background

Double beta decay (ββ) is a family of very rare physical processes with the half-lives at the level of 10^{20} years and above. One of the processes is $\beta^-\beta^-$ decay which is a decay mode of an atomic nucleus in which two neutrons transform into two protons and two electrons with or without two antineutrinos and can be expressed by the following formulas:

$$(A.Z) \rightarrow (A, Z+2) + 2e^- + 2\tilde{\nu} \qquad (2\nu\beta\beta)$$

$$(A, Z) \rightarrow (A, Z+2) + 2e^- \qquad (0\nu\beta\beta),$$

where *(A, Z)* and *(A, Z+2)* are the parent and daughter nucleus, respectively, e^- denotes electron and $\tilde{\nu}$ is antineutrino. Neutrinos are fundamental particles that played a key role in the early universe processes and are important for cosmology and nuclear and particle physics.

Whether the neutrino particle is present or not among the products of the decays described by the formulas above, it takes part in the interaction. Particularly the absence of neutrino in some theoretically predicted modes of ββ decay (so called neutrinoless ββ mode, the second formula) would indicate some special properties of the neutrino particle. The study and detection of these special modes of ββ decay could explain some unresolved questions of neutrino physics, such as lepton number non-conservation or neutrino nature (Majorana or Dirac) [5].

Several large projects concerning this field of physics are ongoing or planned – CUORE [6], GERDA [7], EXO [8], SuperNEMO [9]. Due to the nature of the studied ββ processes (very long half-lives), all running or planned projects follow the similar and tested strategy – maximize the amount of studied material and minimize the background (in general the signals mimicking and covering the studied effect). The background optimization is typically realized using ultra-pure construction materials and a passive shielding surrounding the core of the experimental setup.

The experimental physics group in the Institute of Experimental and Applied Physics of the Czech Technical University in Prague (IEAP CTU) is currently involved in general experiment SuperNEMO and two smaller experiments studying ββ decay – COBRA and TGV. Within these two projects, we study the possibility to utilize pixel detectors. As they were not originally designed for low-level counting experiments, they still need extensive research and development towards understanding of intrinsic background and optimization of technical design. Nevertheless, due to their high-resolution tracking capabilities they could potentially become a very effective tool to classify the measured signals and significantly improve the signal to background ratio. This active background-rejection approach, combined with the standard passive shielding, would allow building an experiment with the sensitivity comparable with much larger and demanding projects.

The data classification and evaluation described further in this thesis were motivated by the needs of real physical experiments. However, cluster analysis applied to data recorded with a pixel detector has much wider area of applications.

2.2 Timepix pixel detector

The description of Timepix detectors is fully covered e.g. in [1] and [2]. The USB interface [10] and Pixelman software [11] have been developed. The Timepix pixel detector creates a frame, which is a 2D image of the same size as the detector composed of pixels. The detector size is usually 256*256 pixels but the chips can be combined into large-area detector.

Figure 1: Timepix detection unit comprising the detector itself and integrated USB readout interface.

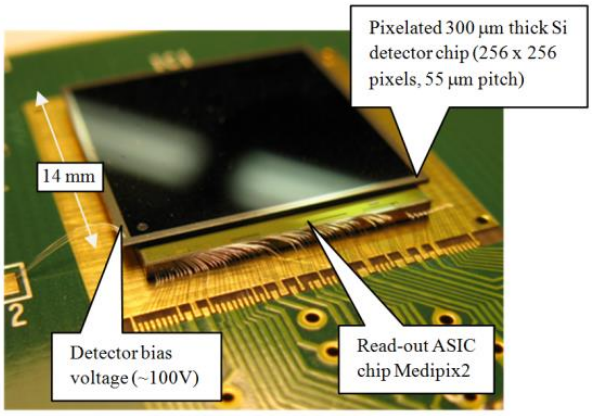


Figure 2: Detail of the Medipix2 / Timepix detector composed of the readout chip and the silicon sensor.

Therefore, the Timepix device output can be expressed as a function $I: t \times (x, y) \rightarrow E;\ (x, y) \in \mathbb{Z}^2, E \in \mathbb{Z} \cap [0, 65535]$, where x and y are coordinates, E is energy deposited within the single detector volume and $t \in \mathbb{R}^2$ is the time when the frame was taken. The zero coordinate (0,0) are in the left upper corner of the image. A cluster denotes a connected component of non-zero pixels. The particle tracks can overlap (creating a single cluster) or the track can be interrupted (creating multiple clusters corresponding to a single particle). In this thesis, it is assumed that a single cluster corresponds to a single track produced by a detected ionizing particle. We usually use 8-connectivity to identify the cluster. Let *C* denotes set of all clusters in time *t* (which corresponds to all particle tracks found in one frame), then

$$C = \{c_i | c_i = \{X \mid I(t, X) \neq 0 \ \& \ c_i \ is \ 8 - connected\}\}.$$

Therefore

$$\bigcup_{c_i \in C} c_i = \{X | I(t, X) \neq 0\}.$$

Each particle type leaves a different type of track. An alpha particle leaves a large and high-energetic blob (Figure 3). This large circular blob is caused by high energy of the alpha particle and by a charge-sharing effect. This is explained in [12]:

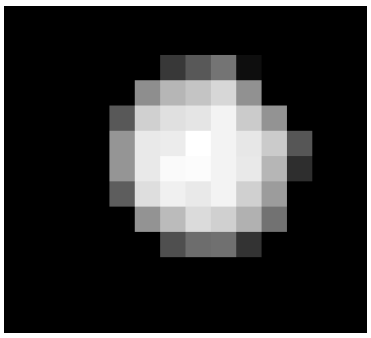


Figure 3: An alpha particle.

Heavy charged particles produce dense carrier tracks in silicon detectors. Under the influence of an electric field, the carriers generated by a particle entering the detector backside, drift towards the corresponding electrode. In the case of a pixellated detecting structure such as Medipix2 and Timepix, charges are shared between pixels giving a signal in cluster of multiple pixels.

An electron (a beta particle) leaves a long curly track (Figure 6) with potentially higher amount of energy at the end due to a Bragg curve. The length and curliness depends on kinetic energy of the particle. A low-energetic beta particle leaves a smaller and curlier track than a high-energetic beta particle.

A photon (gamma particle) is an indirectly ionizing particle. It tends to create a smaller tracks, dot (Figure 4).

Another particle type we want to classify is a muon. Muon particles leave long straight tracks, due to their high kinetic energy and low interaction with detector material (Figure 6).

The last recognized particle type is a proton. This particle is characterized by a medium long thick track with high amount of deposited energy (Figure 7).

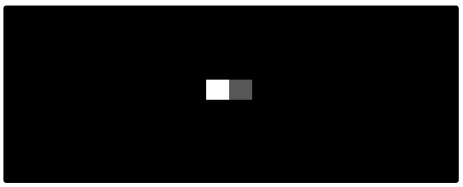


Figure 4: A gamma particle.

Figure 7: A proton particle.

Figure 6: A muon particle.

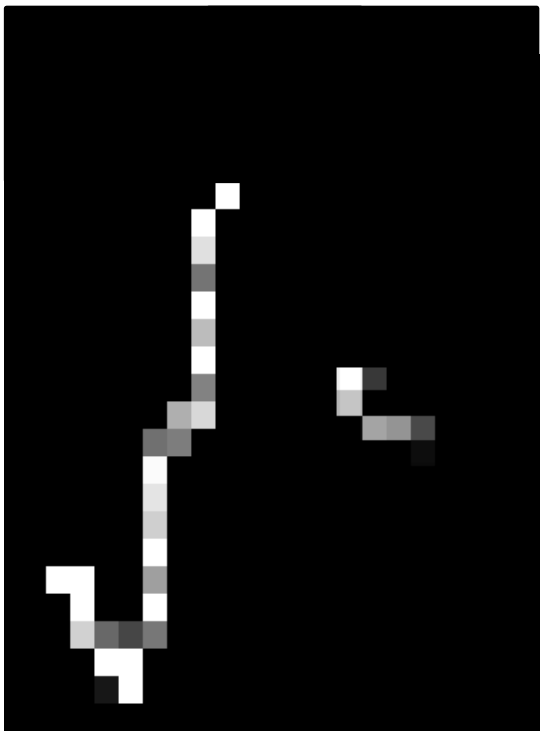


Figure 5: Beta particles.

Each cluster can be described by numbers that depend on its shape, size, energy, etc. We call those numbers a "cluster characteristics". Therefore, the cluster ($c_i \in C$) can be represented as a single vector $f(c_i) = V;\ f: C \to S;\ S \subset \mathbb{R}^N$, where N is a number of used cluster characteristics, f is a function that computes cluster characteristics and S is a whole space, which contains all N-dimensional vectors of characteristics corresponding to all possible clusters. Our aim is to categorize each cluster into a single category with defined efficiency measures (see chapter 2.4 for more details). The categories correspond to the particle types.

2.3 Cluster characteristics

The cluster shape can be described by many parameters. The cluster area is the simplest one. It is a simple number of pixels forming the cluster. Despite its

simplicity, it is a notable criterion to distinguish between alpha blobs and smaller clusters. The area is also the only criterion that can be used to distinguish dots (gamma particles) because they are too small to fit in any more complex criteria.

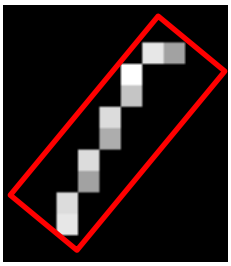


Figure 8: Linearity.

The next criterion, which can be computed easily, is linearity. It is ranged from 0 to 1 whereas one means a straight line and zero is the most non-linear shape. The meaning of "non-linear" depends on the mathematical definition of linearity. We use two definitions. The first is a ratio of width to height of a bounding rectangle that contains the cluster (Figure 8).

$$lin = 1 - \frac{\min(width, height)}{\max(width, height)}.$$

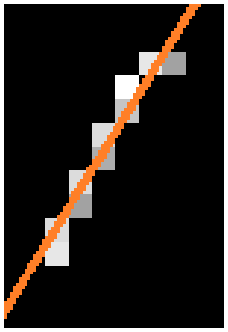


Figure 9: Linear fit.

The second one is defined as a squared deviation from the linear function that is fitted to the cluster. The deviation is normalized into [0, 1] where 1 corresponds to a circle-shaped cluster (Figure 9: Linear fit).

The third characteristic is a tortuosity [13]. It is similar to the linearity but it has more sophisticated definition. For definition of this characteristic, it is necessary to define geodesic length and geodesic diameter [14]. The cluster can be perceived as a polygon P (the pixels are squares of defined size; the actual size does not matter). Let geodesic length $gl(A, B); A, B \in P$ denote a length of a shortest path between points A and B. All points of the path must be contained in the polygon P. Let geodesic diameter $gd(P)$ denote a length of the longest geodetic path in the polygon P.

$$gd(P) = \max_{A,B \in P} gl(A, B) = \max_{A,B \in P} \min_{p=\{A,a_1,a_2,\ldots,B\} \subset P} \sum_{a_{i-1},a_i \in P} |a_i - a_{i-1}|$$

Then the tortuosity of polygon P can be defined as $T(P) = 1 - \frac{|A-B|}{gd(P)}$, where A and B are the same points as in the previous equation (the starting and finishing point of the geodetic diameter path). Because both values are positive and geodesic length is always greater or equal than Euclidean length[1], the tortuosity is ranged from zero to

[1] They are same in a convex polygon. Otherwise the geodesic length is greater.

one. The electron tortuosity is close to one, while the other particles are close to zero (see Figure 10).

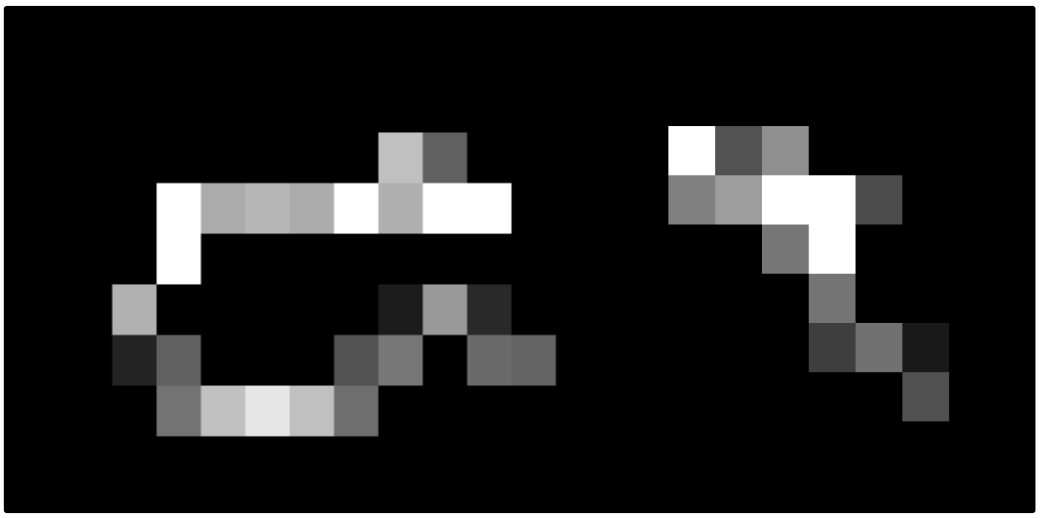


Figure 10: Examples of clusters with tortuosity 0.8 (left) and 0.0 (right).

Figure 11: Convex hull.

The tortuosity might be also defined as a ratio of cluster size to area of convex hull of the cluster (see Figure 11). Nevertheless, this criterion is not used in the thesis because is too dependent on small "branches" of the track, which are not important to the classification and they are probably caused by detector effects (e.g. electronic noise).

Another characteristic is circularity. It is a ratio between the cluster area and a square of the geodesic diameter (the area of a circumscribed circle of the polygon). It is scaled to the disc:

$$circularity(P) = \frac{4A(P)}{\pi gd^2(P)}.$$

The circularity is one for an ideal blob and decreases for elongated objects.

The Ref. [15] suggests another approach to the problem. Instead of defining the characteristic numbers, mathematical morphology operations are used (morphological opening, geodesic dilation, etc.) to filter out particle types one by one. This approach is not used in Pixa software because it is too dependent on the assumptions about the cluster shapes. The method does not classify particles as alpha, beta etc. but as dots, blobs and tracks. It is not sure whether for example the gamma particle always leaves a dot. Moreover, it seemed that the implementation would be complex. In addition, the parameters of the classification (quality of

classification) are defined by parameters of the structuring elements used in the morphological operations. Therefore, quality of classification is not easily adjustable.

2.3.1 Geodesic diameter implementation

The geodesic diameter computation is a well-known problem. The first solution is presented in [16]. Nowadays, there are many algorithms how to compute it, the fastest one introduced in [17] has a linear (relative to number of vertices in the polygon) time complexity. However, those methods works in the simple polygons (with no holes) only. It is possible that a particle track has one or more holes. This problem of finding a geodesic diameter in polygon with holes is solved in [18]. This method has a $O(n^{7.73})$ or $O(n^7(log\ n\ +\ h))$ time complexity, where n is number of vertices and h is a number of holes. This worst-case time is too high for our problem – the typical track polygon has many vertices. Also, it would be hard to implement such an algorithm. The most efficient approach seems to be described in [13], it is based on approximate but fast computation.

Therefore, the simplest approach has been chosen. We know that the track is composed of pixels of the same size. Therefore, we can use an algorithm that is similar to flood fill: the algorithm computes geodesic length between some pixel and the rest of pixels. The lengths are computed in waves spreading from the beginning point. The starting points are chosen as the ends of the morphological skeleton of the particle track. The average and worst time complexity is $O(p * e)$, where p is a number of pixels and e is numbers of ends of the morphological skeleton. Because there are not many such ends, it runs nearly in linear time. In addition, it has a small multiplicative constant. The clusters are usually small (tens of pixels), therefore, this approach is efficient for this problem.

2.4 Cluster type recognition methods

There are many methods and algorithms to classify a cluster represented by an N-dimensional vector. The two-class classification divides the space of all possible values into two parts – the signal events and the background events. The signal events are the events we want to select. The background events are the rest. The multiclass classification divides the space into more parts – one for each category. The multiclass classification for M categories can be emulated by performing (M-1)

two-class classification by choosing one category as a signal and the rest as a background.

The quality of classification is defined by the parameters: efficiency and background rejection. Efficiency is a ratio of number of correctly classified signal clusters to the number of all signal clusters. Background rejection is a ratio of number of correctly classified background clusters to a number of all background clusters.

The easiest approach is to define thresholds for each component of the vector (of characteristic numbers) by constant ranges, which are independent of each other. Mathematically speaking, it selects an N-dimensional rectangle out of subspace of all possible values (which is a subspace of $\mathbb{R}^N$). This approach is used in cluster classification plugin (BSP) of the Pixelman software [11]. This approach is simple to implement and it is very fast. Another advantage is that it does not require any learning data. However, this can be considered as a disadvantage because it depends on the user's observation and experience instead of real data. Another disadvantage lies in a low number of degrees of freedom of the N-dimensional rectangle definition.

2.4.1 Linear discriminant analysis

Linear discriminant analysis (LDA; also known as Fisher's linear discriminant) [19] is a statistical method using a supervised learning. It divides the N-dimensional space of all possible values into two parts (two categories) by a hyperplane (this is why it is called linear). It is based on projection of the N-dimensional space into a single dimensional space and defining the probability density functions (PDF; one for signal, one for background) on the single dimensional space. The projection and PDFs (equivalent to the definition of hyperplane) are computed from the learning data. It assumes that both PDFs are normally distributed. This approach can be generalized by splitting the space of all values by non-linear hyperplane, defined by a function. This is called Function discriminant analysis. In Pixa, the LDA is one of used approaches although it does not perform well due to low amount of degrees of freedom (the only degree of freedom is the hyperplane).

2.4.2 Decision tree

The decision tree [20] is also a supervised learning method. It creates a binary decision tree. The root node corresponds to the whole space. Then (during the learning process) the space is divided into two parts by a criterion (usually the one that splits the space into the most distinguishable parts) and each part corresponds to a new child node. This process is performed recursively until the separation is good enough or the node contains too few events. Therefore, the space is divided into differently sized N-dimensional rectangles optimally. There are several ways how to enhance the classifier. We will describe three of them – adaptive boosting, gradient boosting and bagging.

Adaptive boosting is based on creating multiple decision trees and event weighting. Each event is assigned a weight that depends on the success of the classification of the event in the previous trained decision tree (during the evaluation phase of the training). The misclassified events in the previous tree get higher weight. The weights of all events in the dataset are normalized so that sum of the weights is constant. The predictions from all trees are combined into a boosted event classification function:

$$y_{boost}(x) = \frac{1}{N_{collection}} * \sum_{i}^{N_{collection}} \ln(\alpha_i) * h_i(x)$$

where h_i is the individual decision tree and α_i is the weight.

Gradient boosting is based on simple additive expansion and error function minimization. The model is characterized by a function F(x), which is a weighted sum of parameterized decision trees $f(x, a_m)$:

$$F(x) = \sum_{m=1}^{M} \beta_m f(x, a_m)$$

where β_m are expansion coefficients. The a_m, β_m are computed by minimizing the deviation from the true value y defined by a loss-function (this function can be defined in different ways, e.g. squared error loss $L(F, y) = (F(x) - y)^2$; the TMVA package uses the following [21]):

$$L(F, y) = \ln\left(1 + e^{-2F(x)y}\right).$$

Bootstrap aggregating (bagging) is a technique where the decision tree (or other classifier) is repeatedly trained using resampled training data set. The resulting classifier is an average of the individual decision trees. This technique helps to stabilize the classification procedure.

2.4.3 Neural networks

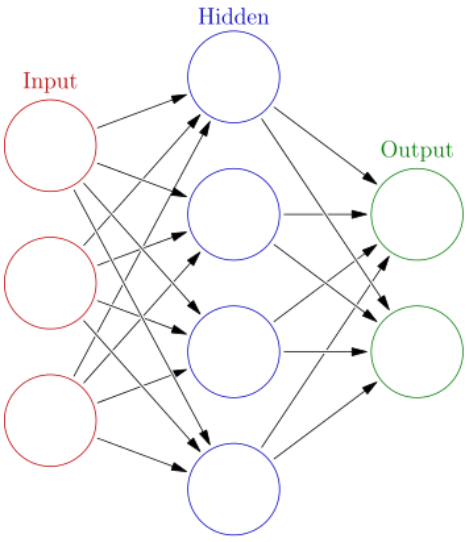


Figure 12: Example of MLP [33].

A multilayer perceptron (MLP) [22] [23] is a feed forward artificial neural network (ANN). ANN is a mathematical model consisting of a group of interconnected artificial neurons. Each neuron has many inputs and one output. Mathematically it is a function $f(x) = K\big(c(x)\big)$, where c is a composition function, which composes all inputs into one number (e.g. nonlinear weighted sum $c(x) = \sum_i w_i g_i(x)$, where w_i are weights computed by learning process). The $K: \mathbb{R} \to [-1,1]$ is called an activation function (e.g. a hyperbolic tangent or a sigmoid). The network learning is based on minimization of an error function. There are many algorithms to perform the supervised learning, e.g. back-propagation (which is used in this thesis), BFGS or genetic algorithms. It is also possible to use an unsupervised or reinforcement (the data are generated by agent's interaction with the environment) learning paradigms. However, it is not used in this thesis because we need to assign tracks to categories based on particle physics. MLP neurons are structured into layers. There is a single input layer, zero or many hidden layers and a single output layer. Each neuron reads (as the input) an output of all neurons in previous layer – except for the input layer. The MLP divides the space into many subspaces whose shape depends on the learned data and network structure.

2.4.4 Support vector machine

The Support Vector Machine (SVM) [24] is a non-probabilistic binary linear classifier, which tries to find a hyperplane that divides the N-dimensional space (of all possible values) into two halves. This model is extended to nonlinear classification by defining a projection that transforms the original input space into high-dimensional space, where it is possible to divide the signal and background sets by a hyperplane [25]. It is called a kernel trick. Therefore, the SVM algorithm divides the space into two subspaces by arbitrary-defined (N-1)-dimensional function.

2.4.5 Unsupervised learning

There are also many types of classification or learning algorithms that do not require a learning data set (unsupervised learning). However, this is not useful for this thesis, because the classification problem, stated in chapter 2, requires categorizing the tracks into parts that corresponds to particle types, which are defined by particle physics.

2.5 Calibration

The energy deposited in a pixel is measured in dimensionless values called channels. Therefore, it is needed to calibrate them into standard energy units (keV). This is performed by measuring well-known emitters [26] and fitting measured values into a so-called calibration function. This function projects keV-valued units into measured channels. It was estimated (also in [26]) to:

$$f(x) = ax + b - \frac{c}{x - t},$$

where *a, b, c* and *t* are the parameters. We need to define an inversion function to compute a keV values from channels:

$$h(x) = -b + x + a * t$$

$$f^{-1}(x) = h(x) + \frac{\sqrt{\left(h^2(x) - 4a * (-c - bt + xt)\right)}}{2a}.$$

2.6 Software technologies

The first choice in software development implementation is to choose the programming language and the runtime. Nowadays, there are three main statically typed objective programming languages – C++, C# and Java. The C++ was chosen

for many reasons. The first reason is performance – the data processing required manipulation with large amounts of data. Therefore, the managed languages are not a good option, because they bring additional overhead with memory allocation. In addition, C++ compiler (Visual C++ or Intel C++) implements better optimizations then C# or Java. There are many benchmarks (e.g. [27]) proving that C++ outperforms C# or Java. The Visual C++ compiler was chosen because of its good performance and perfect integration in the Visual Studio IDE and it is available free of charge.

The choice of the mathematical and statistical framework was simpler. The ROOT framework [28] is considered as a standard in particle physics. It contains many routines and algorithms that are useful in this thesis. For example, it contains the TMVA package [21], which handles all classification methods used in this thesis. The ROOT and TMVA API are fully objective and compatible with C++. The library can store all generated outputs (graphs, histograms …) in a special file format called ROOT file. This file format is also widely used in particle physics. Both TMVA and ROOT work under Linux, Mac OS and Windows.

There are many major graphical user interface (GUI) libraries for Windows, from which we evaluated the Qt toolkit and MFC. Both of them are based on native Windows controls, ensuring consistency of the user inteface (UI) in the user's system. Both are also integrated in the Visual Studio IDE, although MFC has a tighter integration. In the end, the MFC was chosen because of the more sophisticated integration in the IDE and much better personal experience. This choice causes loss of cross-platform. Therefore, a new Qt-based version is under development. This version will be available for the Linux and Mac OS system as well.

3 Implementation

In order to implement required classification algorithms and to process data (both simulated and measured) the software called Pixa was developed. The application is divided into layers (or modules). The layers are partially ordered in order to simplify application design (and its understanding by other people) and to eliminate unexpected dependencies between parts of the code. In addition, it increases the reusability of the code. Each higher layer use services from lower layers only (the external libraries and system API are considered as lowest layers).

There are four basic layers:

1. Data sources layer
2. Processing tasks layer
3. Data processing layer
4. User interface layer.

In addition, there is a special "utility" layer that contains various helper and infrastructure routines and classes. The layers and their dependencies are illustrated in Figure 13: Layer dependencies.

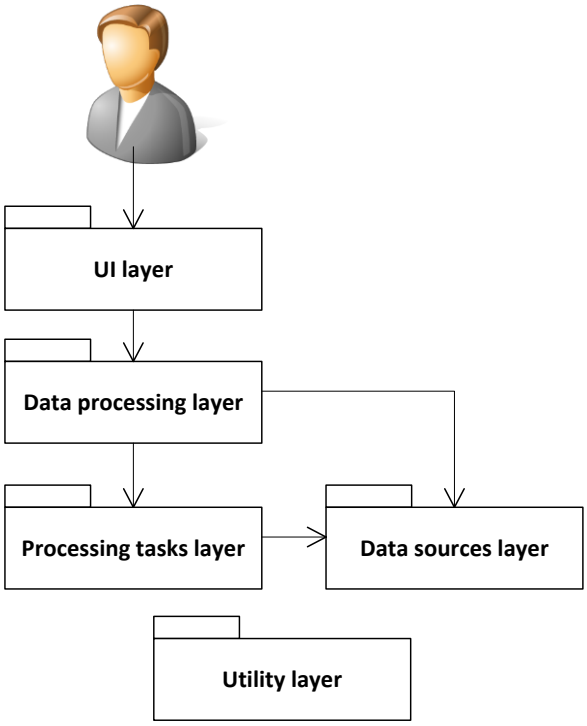


Figure 13: Layer dependencies.

The data sources layer contains classes that are responsible for reading and simple pre-processing of measured or configuration data. The processing tasks layer contains routines and algorithms for data processing and classification. It also contains cluster classification routines as a sub-layer. The data processing layer provides a consistent runner that manages all data sources and processing tasks and their execution. UI layer is responsible for user interface shown to user.

3.1 Formats of input data

The application can process many formats used in the community working with the Timepix detector to store measured frames and new formats can be easily added to the application.

3.1.1 Text sparse matrix

The simplest format to keep measured frames is text sparse matrix. It is a simple text file. On each row there are three numbers – the X and Y coordinate of affected pixel and the deposited energy. The deposited energy is stored in "bins" – a dimensionless number that describes energy deposited in the pixel. We can use calibration data to compute a real energy in eV[2]. Only the affected pixels are stored in the file. The frames are separated by a line containing only a "#" symbol. If there are no affected pixels in the frame, the frame is empty (but the "#" symbol separating the frames is still present). It might be a single file or a folder of files with frames with this format.

3.1.2 "Cluster log" format

The cluster log is also a text format. It contains affected pixels but they are grouped by clusters (see chapter 2). Each frame is introduced with a line in following format: "Frame <frame number>". Optionally there can be some meta information about the frame – a start time and an exposure time. The C format string describing the frame header: `%s %d (Start time = %lf, Exposure = %lf s)`. After the header, there are one or more lines – one for each cluster. In the line, there is a list of the activated pixels in the cluster. Each pixel is represented by three numbers – X

[2] Electronvolt is a unit of energy equal to approximately 1.6×10^{-19} J. By definition, it is the amount of energy gained (or lost) by the charge of a single electron moved across an electric potential difference of one volt.

and Y coordinates and the deposited energy (see 3.1.1) – in square brackets. The sample frame with two clusters might look like:

```
Frame 20 (Start time = 1340733529.9247336 s, Exposure = 1 s)
[211, 35, 1] [212, 35, 46] [213, 35, 86]
[73, 181, 3] [74, 181, 30] [75, 181, 43] [76, 181, 5]
```

The cluster logs can be in a single file or as a folder with multiple cluster log files.

3.1.3 Raw images

The frames can be also saved as a raw (do not mistake them for a RAW format used in digital cameras) image data. It is simply uncompressed image without any header, there are just binary codes of the colors of the pixels (we use only grayscale images, so each pixel is represented by a single byte (0-255; 255 shades of grey). The raw image has the same size as the detector and the brightness of the pixel corresponds to deposited energy in the pixel.

3.1.4 Pixelmask

The pixelmask format is used to keep a mask expressing locations of defective or noisy pixels in the detector. It is a text non-sparse matrix – the values are separated by a space and each matrix row is on a single line. The value can be either "0" or "1". "1" means that the pixel is OK, the "0" means that the pixel is defective. The matrix has the same size as the detector.

3.1.5 Calibration matrix

The calibration matrix is used to keep values for computing real (calibrated) energy values from dimensionless bin values. The calibration process is described in chapter 2.5. There must be 4 files – each for one parameter (a, b, c, t). The format is similar to pixelmask (see 3.1.4) but there are decimal values instead of zeroes and ones. Each value in the matrix means value of the corresponding calibration parameter.

3.2 Data Sources Layer

This layer contains classes called data sources (Figure 14). Data source is a class that is responsible for reading data from the disk. Then it passes the data to higher layer. Each data source class must be derived from `DataStorage` class which

defines interface for all data sources and some helper methods (e.g. for loading matrices saved as text files, which are used in the community working with the Timepix detector). Each data source class that reads measured frame data must be derived from `MeasurementDataStorage`. This base class provides functionality for reading measured frame data:

- Basic raw and prefiltered data statistics
- Many properties that are common for any Timepix data (e.g. detector pitch, definition of multidetectors, etc.)
- Support for masking "wrong" pixels on the detector
- Support for calibration

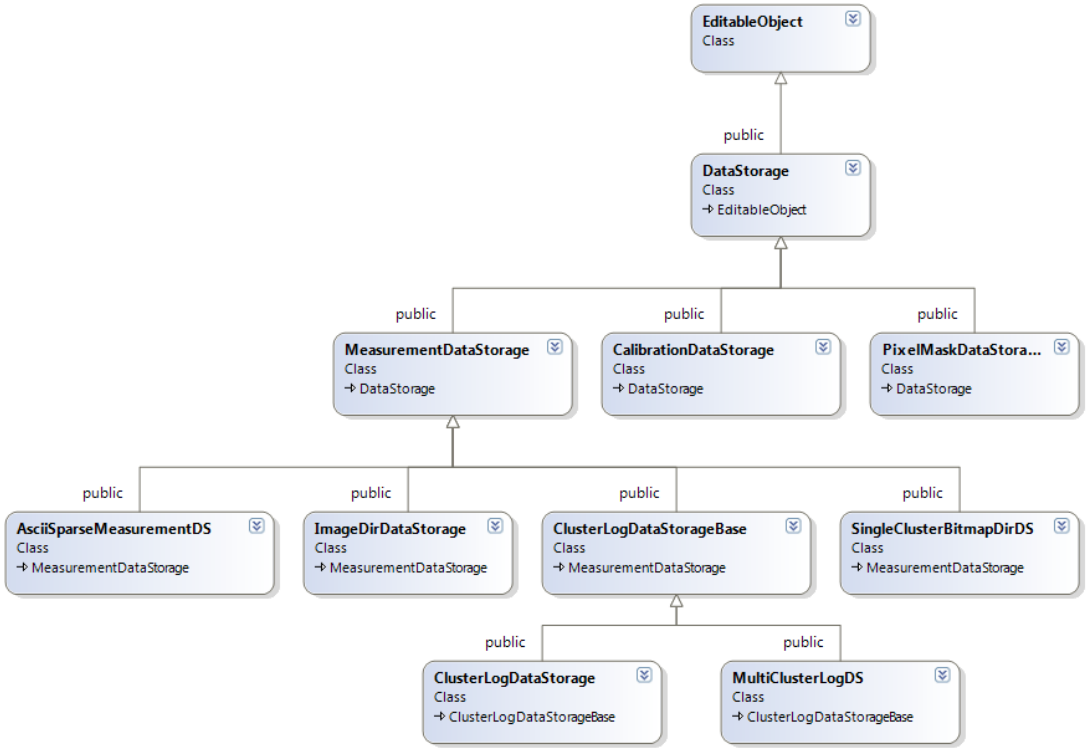


Figure 14: Data sources class diagram.

There are two non-measurement data storages. The first one is called `PixelMaskDataStorage` and it is used for loading pixelmasks (see 3.1.4). The second is `CalibrationDataStorage` and it is responsible for loading calibration matrices (see 3.1.5).

In addition, there are several measurement data storages:

`AsciiSparseMeasurementDS` is a data storage used for reading text sparse matrices (see 3.1.1). Besides file reading capabilities, it also contains routines for searching clusters in frames. The user can choose whether he wants to use 4-

connectivity or 8-connectivity. The cluster searching routine is implemented as a variation of a queue-based flood fill algorithm.

`ClusterLogDataStorageBase` (inherited by `ClusterLogDataStorage` and `MultiClusterLogDS`) implements reading cluster log files (see 3.1.2). The cluster searching functionality is not needed here because the pixels are divided into cluster in the file already. The descendants implement opening the files only (a single file or all files in the folder).

3.3 Processing Tasks Layer

This layer (Figure 15) is the working core of the application. It contains all the executive code that performs all the analyses and cluster classification. The layer consists of two main parts: 1) all tasks and 2) cluster type classification routines. The task is a class that is a part of the processing chain, takes the measured data frame by frame, and processes them. Each task can do three main things with the frame: firstly, it can throw away whole frame, so the frame will not take part in further processing. Secondly, it can modify the frame (e.g. throw away clusters, apply some data transformation etc.). Alternatively, it can analyze the frame and produce some (e.g. statistical result). Each task can do all (or none) of these three things, but it is strictly advised that tasks are divided into:

1. filtration tasks, which throws away frames and/or clusters to remove them from further processing,
2. data transformation tasks, which applies some transformation to the data, or
3. analysis tasks, which do not modify the data and performs some analysis.

Each task that wants to participate in processing chain must implement at least `PipelineTask`. This abstract class defines some basic methods that are called at the important moments of the data processing. However, all implemented tasks are inherited from higher class `ProcessingTask` that allows the task to receive the data frame by frame and participate in the data processing. The most important methods defined in the `ProcessingTask` class are `FramePassedThrough` (that decides whether the frame is passed to the next task in the processing chain) and `FrameProcess` (that receives single frame and perform the actual processing).

Each task must register itself (at the application initialization phase) and its user-editable properties (in the constructor) in Object and property registration

system (see 3.6.1). Otherwise it will not be visible in the GUI and it cannot be (de)serialized.

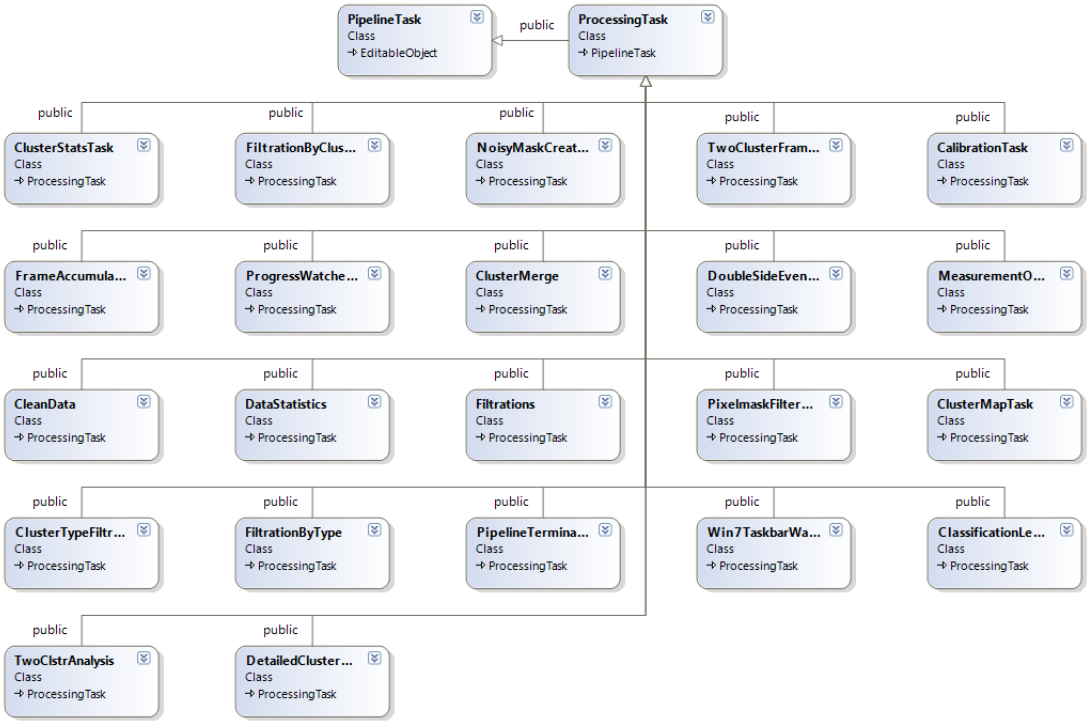


Figure 15: Task classes.

The cluster classification consists of two phases. Firstly, the cluster is analyzed and its characteristics are computed. The computation is implemented in the `MpxCluster` class that holds both pixel data and computed characteristics. The computation is realized by calling the `DoAnalysis` method. This method checks whether the analysis of the cluster properties has been made and then it computes all the characteristics (the characteristics are described in chapter 2.3). Those numbers are very important in the cluster classification.

The cluster classification is implemented in the `ClusterClassification` class. At the beginning of the processing, it loads data that was created by the learning task. There can be multiple sets of learned data; the user can choose which data and which classification method will be used. All learned data are stored in a ROOT file that is distributed with the application. Each type of detector behaves differently therefore, a user can create his own set of learned data. Each classification method has its own format. They are stored as a XML string. The classification core objects are constructed using the XML string specific to the selected method. The classification methods themselves are used from TMVA [21], which is a library for

machine learning environment for the processing and parallel evaluation of multivariate classification. This library is included in the ROOT [28] package.

Each cluster is represented as a vector $v \in \mathbb{R}^N$, where N is a number of cluster characteristics (computed by MpxCluster class). Because all classification methods implement the same interface (TMVA::MethodBase) the only thing we need to care about is whether selected method supports multiclass evaluation or not. Nevertheless, we can simulate multiclass evaluation (for methods that do not support multiclass evaluation) by performing multiple learning/classification runs – one for each type of the particle. We can treat one particle type as a signal and the other particle types as a background. Therefore, the classification process simply reads previously computed cluster characteristics and passes them to the TMVA routines (with regard to multiclass support). Then the resulting $MvaValue \in [-1,1]$ is compared to the MVA value threshold given by the user.

3.3.1 Statistical tasks

The first described group of tasks are statistical tasks. Those tasks do not modify the data at all, they only scan the data and perform basic characterization of the data set.

The "Data statistics" (DataStatistics) task implements creation of histograms that describe basic data properties, e.g., histogram of distribution of cluster sizes, distribution and history of energy deposited energies per cluster or frame etc.

The "Two cluster frame analysis task" analyzes frames with exactly two clusters. In addition to statistics similar to the previous task, it computes distribution of cluster distance. This task has been created to analyze data from the setup composed of multiple detectors (such as TGV experiment, where detectable signal is two X-rays in coincidence with energy ~21 keV each).

The "Cluster statistics" task analyzes the cluster characteristics – their distribution and whether they are significant or not.

3.3.2 Other processing tasks

This group contains tasks that perform an analysis of measured frames. Contrary to the previous group, these tasks provide some other output than histograms.

The “Noisy mask creator”, as its name implies, is used to create so-called noisymasks. Noisymask is a special type of pixel mask that is generated according to the number of counts recorded in a single pixel during the whole measurement. We assume that a pixel that is “hit” too many times comparing with the average is defective (noisy). The threshold for being noisy is specified by an operator using the histogram of pixel activity.

The “Cluster processing” task serves two main purposes. Firstly, it can take the data from the processing chain (to perform filtering, calibration, etc.) and write it onto the disk as a cluster log (see 3.1.2) so that they can be used later or further processed by another software. Alternatively, it can be used for manual checking of small amounts of data. Secondly, it can visualize each frame at the screen in a separate dialog. It allows user to see the shape and energy of the cluster and the cluster characteristics. All frames are held in the memory therefore the user can return to any frame. The example pictures of clusters in this thesis are made with this functionality.

The “Frame accumulator” task simply reads all frames and superimposes them it in a single frame. Thus, it creates a heat map of each pixel activity in the detector. The heat map is stored as a 2D histogram.

The “Cluster map” task is similar to the previous one – it computes the centroid of each cluster and writes it into a heat map.

3.3.3 Data transformation tasks

The purpose of these tasks is to apply a transformation to the frame.

The “Cluster merge” task merges all clusters in a single frame in one big cluster. However, due to detector defects, the pixels may not connected to each other. The pixels in such cluster may not be connected. It can be used for a special scenario when we process data that are known to be a single cluster.

The “Pixelmask filter” task’s purpose is to apply additional pixelmask in the middle of the processing chain or as a task-local filter.

3.3.4 Filtration tasks

The filtration tasks analyze frames and clusters according to some criteria and remove those that do not meet the defined requirements.

The general filtration task allows user to remove clusters and frames according to simple criteria. The clusters can be filtered according to the cluster energy, cluster

size, cluster energy amplitude and some special criteria specific to the TGV experiment. To filter the, one can use: number of clusters in the frame, size of the largest cluster and distance of the clusters in two-cluster frames. In order to let the cluster or frame pass, all selected criteria must be met. A user can negate the filter, which means: keep only the frames or clusters that do not meet at least one criterion.

The “Filtrations by cluster properties” task is a variation of previous task – it filters out clusters according to selected cluster properties used for cluster type recognition.

The “Filtration by cluster type” task accepts only clusters that are recognized as a specified cluster type (alpha, electron, etc.) with given probability (MVA value).

3.3.5 System (infrastructure) tasks

The system tasks are not available to the user (with one exception), they function as a support for the processing chain.

The ProgressWatcherTask and Win7TaskbarWatcherTask are very similar. Both of them monitors the processing and send the progress to the GUI so that the user can see how many frames were processed so far. The first one sends a custom Windows Message (WM_PROGRESS, WM_STARTED and WM_FINISHED) to the main view (the content of the main window) and is essential for the progress bar. The second one works only on Windows 7 or later and it is responsible for sending the same Windows Messages to the main window frame (the details are discussed in chapter 3.5) which creates and maintains the progress bar in the Windows taskbar.

The “Pipeline terminator” task is the only one available to the user. It stops the processing chain by throwing away all frames so that any other task after this one is not invoked.

3.3.6 The learning task

The learning task is one of the most important tasks in the software. It reads separately data sets for each particle group and performs the supervised learning. It is a special type of task doing all the work in the EndProcessing stage.

At the beginning, it loads all the data from each particle group data set. The data are specified as pointers to the measurement data sources therefore it can process any format of the data. Then, it initializes the TMVA objects, fills them with the data, and starts the learning process (implemented in the TMVA). In addition, it creates graphs of efficiency and background rejection (see 2.4 for definition). Those

graphs are created in the following way: for each MVA value in the range of [-1, 1] with some small step (0.01) we evaluate how many particles were correctly classified as signal and background. Then, we combine those values into one graph: for each MVA value, we make a point in the graph, where x=efficiency and y=background rejection (see chapter 4.2 for examples of this graph).

3.4 Data Processing Layer

The data processing layer provides a consistent runner that manages all data sources and processing tasks and their execution.

The layer consists of a processing context that contains one or more processing chains. The possibility of having more processing chains that are executed in parallel is implemented in this layer but it was not implemented into the GUI. Therefore, there is always a single processing chain. The context creates a single thread (called a dispatcher thread) for initialization of data storage objects and pipelines. The data storage objects are initialized in the order that all storages that depend on any other storage will be initialized after such objects. For example, if the user adds a clusterlog storage and a storage with pixelmask (the clusterlog storage depends on the pixelmask, because it uses it), the pixelmask storage would be initialized before the clusterlog storage. Then it starts another thread for each processing chain. The shepherd thread waits for all processing chains to finish and it finalizes the data storages. The context is responsible for serializing and deserializing all settings and tasks and data storages. The serialized settings are stored in custom XML format and it uses the property registration system (see 3.6.1), which is a simple substitution of the reflection concept, which can be found in higher languages (e.g. Java or C#).

The processing chain manages all the tasks and the data flow. The processing chain is always bound to a measurement data storage, called the primary storage, that the measured frames are taken from. After the initialization of the tasks, it loads frame by frame from the initialization storage. Each frame is processes by all enabled tasks in the chain in the order defined by the user. The task can modify the frame (or throw it away completely), therefore n-th task always gets the result of (n-1)-th task (except for the first one). In addition to this, the user can apply special filtering tasks, called the task-local filters. Those filters are normal tasks that are enabled for use for such a purpose. Let n-th task has two task-local filters TLF1 and TLF2. Therefore the output of (n-1)-th task is processed by TLF1, then by TLF2 and finally by n-th task

(which gets the output of TLF2). The output of n-th task is not used for further processing (the task-local filter and the task work on a copy of the measured frame), therefore (n+1)-th task uses the output of (n-1)-th task. This example is illustrated by Figure 16. Each task can have three task-local filters (this limitation is for the sake of GUI simplicity). The whole processing is illustrated by Figure 17.

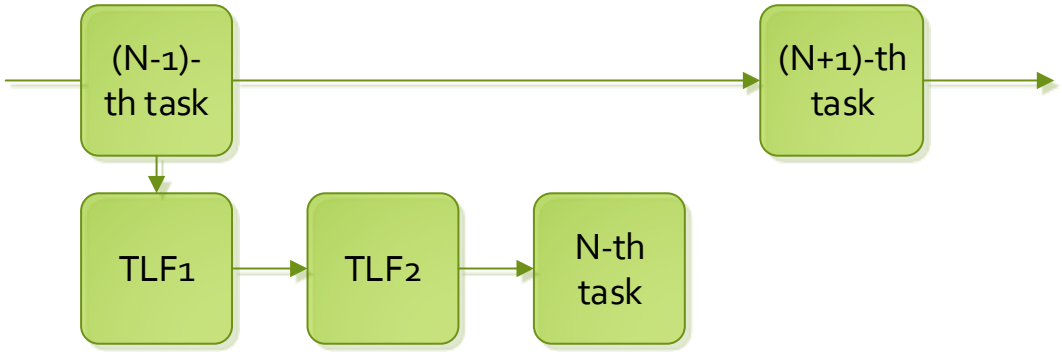


Figure 16: Task-local filter processing example.

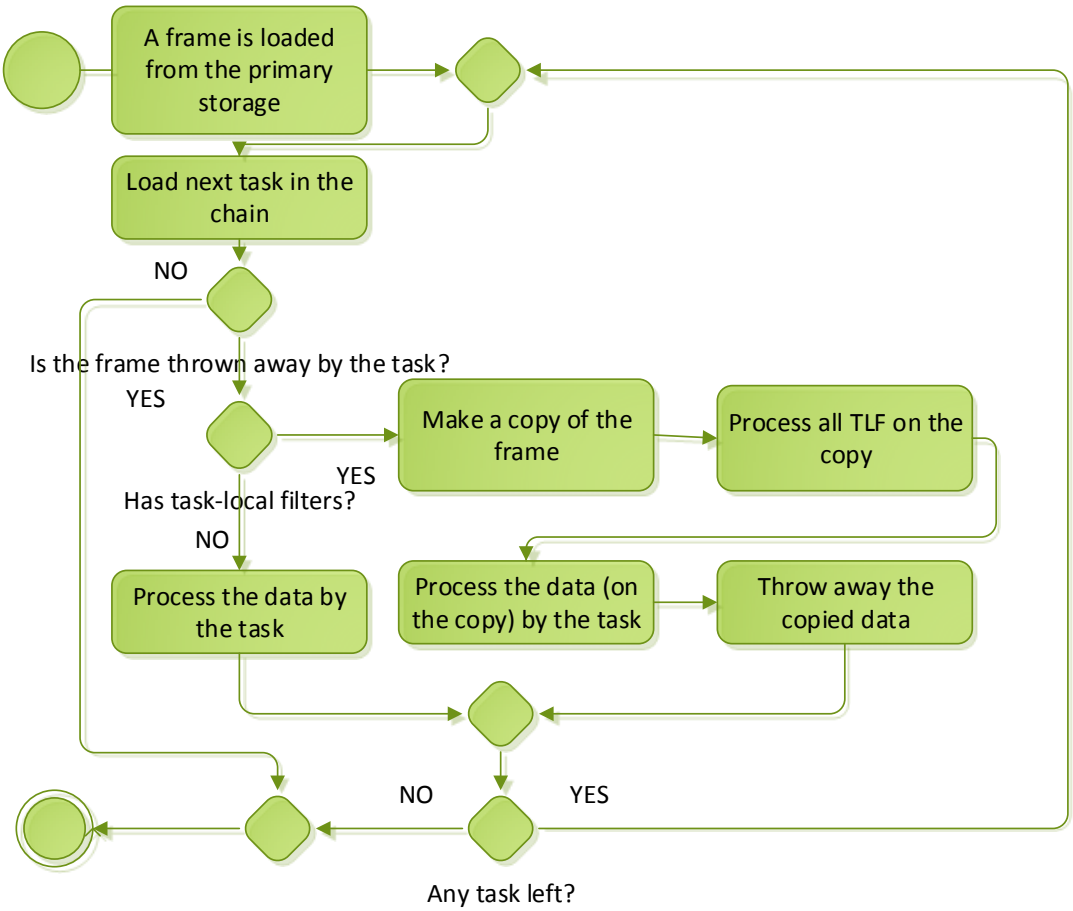


Figure 17: UML Action diagram of processing a single frame.

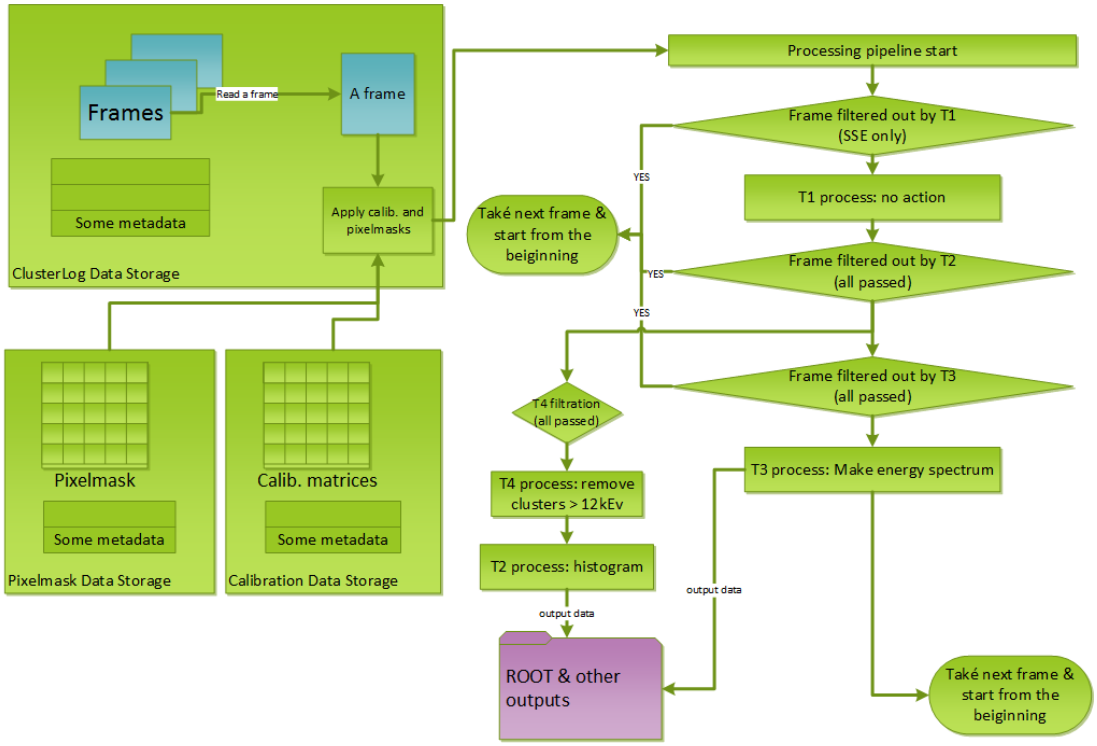


Figure 18: The processing example.

Let us have an example: There are three data storages – one for measurement data in ClusterLog format, one pixelmask and one calibration. In addition, it contains four tasks. T1 filters out all frames that are not SSE frames but does not do any processing. T2 makes a histogram (output data) and it has task T4 assigned to it as a task-local filter, which removes all clusters with cluster energy ≤ 12 keV. Because of the T4, the T2 creates a histogram of all SSE frames (from T1) and clusters with cluster energy ≤ 12keV. T3 makes a histogram of whole energy spectrum. Unlike the T2, the T3 processes all SSE frames, cluster energy is irrelevant in this case (Figure 18).

3.5 User Interface Layer

The GUI is based on the MFC library. This library was used because of its good integration with Visual Studio. The MFC library UI is based on the Document/View architecture [29]. There are three classes for each window:

1. The Document class, which represents the model or data. The Pixa document is very simple. It contains the processing context and supports the save/load operations, which are implemented by the processing context's serialization capabilities.

2. The View class, which represents visualization of the Document. In Pixa, it means showing UI for modifying processing context, tasks and datasource properties. The tasks' and data sources' properties are modified by the property grid (UI component), which is part of MFC. The property grid is slightly modified to support all features we needed so it can be integrated with the property registration system easily.
3. The Frame class, which represents the whole window. It contains menu, toolbars and the View of current Document.

In general, there can be more Documents and Classes, but there is only one in Pixa. The support dialogs are either part of the Win32 API (save/open dialog) or a simple dialog, which does not contain documents or views.

3.6 Utility Layer

The utility layer contains the object and property registrations and many small methods and classes that implement common routines like format conversions, system calls etc.

3.6.1 Object and property registration system

The object and property registration system was created to substitute reflection and attributes, which can be found in higher languages like C# or Java. It is used to read and modify object's properties (e.g. settings path to files, tasks' processing options …) dynamically and to locate and construct the objects.

Each object, which supports this, must be inherited from the `EditableObject` class, it has to register each property (with some metadata) in the object's constructor and it has to contain a few members, which are easily created with a preprocessor macro. The property is a simple instance field of the object. It can be either a simple type (int, bool, double …), a C++ string (`std::string`) or pointer to another editable object. Each object is assigned an application-wide unique integer that acts as an identification of the object and it is used for referencing the object in persistent data location (XML file). In addition to property metadata, the `EditableObject` implements the (de)serialization of all registered properties. The deserialization has two phases: Firstly, it constructs the objects and deserializes simple and string properties. Secondly, it deserializes all pointers.

A task property is a metadata object (derived from `TaskPropertyBase`). It holds a name, a description, a member pointer (location of the referenced field in the object) and a type of the property. Also, it has abstract methods for getting and setting the value. There are two types of task properties: simple and pointer. Simple type property is a templated reference to non-pointer type like int or string. The string and double have a specialized template. An extension can be added to the simple type property. The extension can contain additional metadata about the property. Syntactically, it is a plain class that is "added" to a task property class with multiple inheritance. There are three extensions: enumeration, folder path and file path. The extended property grid knows those extensions and uses them for the GUI. Pointer type property is a templated reference to the pointer of editable object. The value setter is restricted by the pointer's type or by a custom method.

The classes itself are registered too. There is a singleton object (`ObjectRegistration`) that holds metadata (names and a factory method) of each class. The class is registered automatically by a code in the constructor of a static member of the class. The constructor is guaranteed to be automatically executed during the initialization of the software. There are few macros to simplify an implementation of the required code.

4 Results

This chapter describes experiments performed to evaluate the quality of classification (efficiency and background rejection parameters, defined in 2.4). Both values are visualized in a single graph – because high efficiency comes with low background rejection and vice versa. Unless stated otherwise, the graphs are created using a data where we know the type of particle that created the track: we evaluate the efficiency and background rejection for the MVA values (ranging from -1 to 1, with a step of 0.01) and then put the point into the graph. See chapter 3.3.6 for better explanation.

4.1 Testing data sets

To evaluate the results of the methods used in this thesis two data sets were used. Data sets were provided for five types of particles – alphas, gammas, protons, muons and electrons.

4.1.1 Data for the learning the method

Simulated (for electrons, photons and muons) and experimental data (for alphas and protons) were divided into two groups. The first group was used to learn the methods included in Pixa software. The second one served as an evaluation data set. The efficiency/background rejection graphs presented in this chapter were constructed from this set.

The alpha particles data was retrieved a real measurement (detection of radon progenies from the air) due to the imperfect model of charge sharing effect included in the simulation model. Their energy spectrum and example of frame measured for 100s can be found on Figure 19.

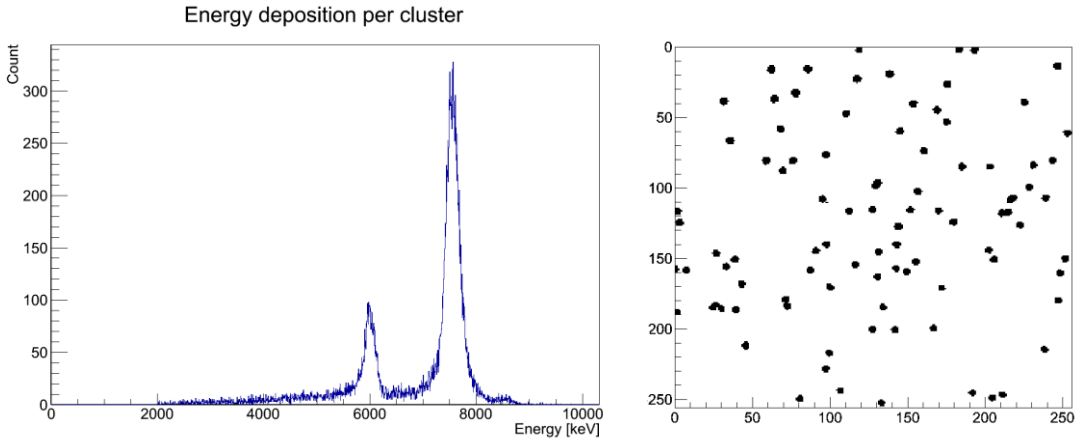


Figure 19: Alpha particle characteristics (left: energy spectrum of alpha particles, right: shape of detected particles).

The electron signals were simulated to correspond to the electrons produced by the double beta decay process in ^{116}Cd nucleus (two electrons with sum energy ~2.8 MeV). The energies of these electrons are continuously distributed in the range up to 2.8 MeV (Figure 20).

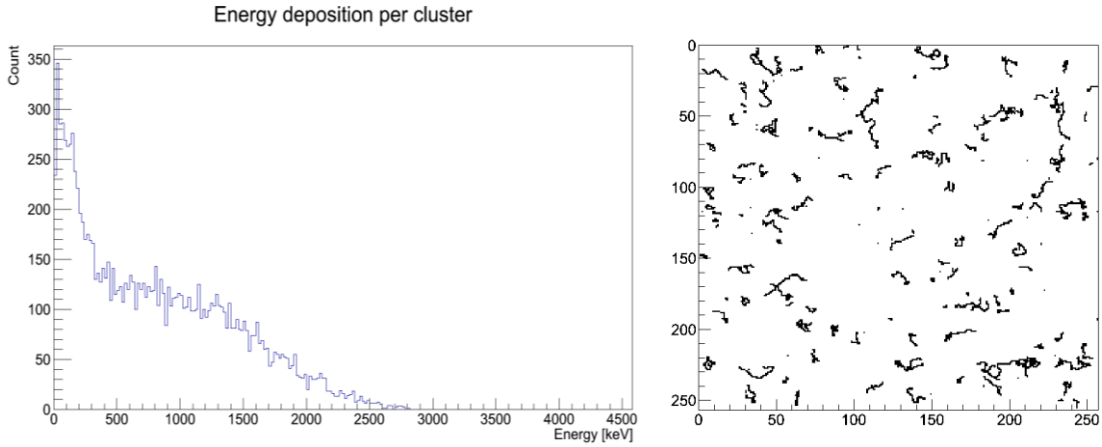


Figure 20: Simulated electrons characteristics (left: energy spectrum of electrons, right: shape of detected particles).

The gamma particles were simulated as a monoenergetic with energy 2.6 MeV. Their detected energy spectrum is continuously distributed because the interaction of gamma particles with the detector are rather complex (Figure 21).

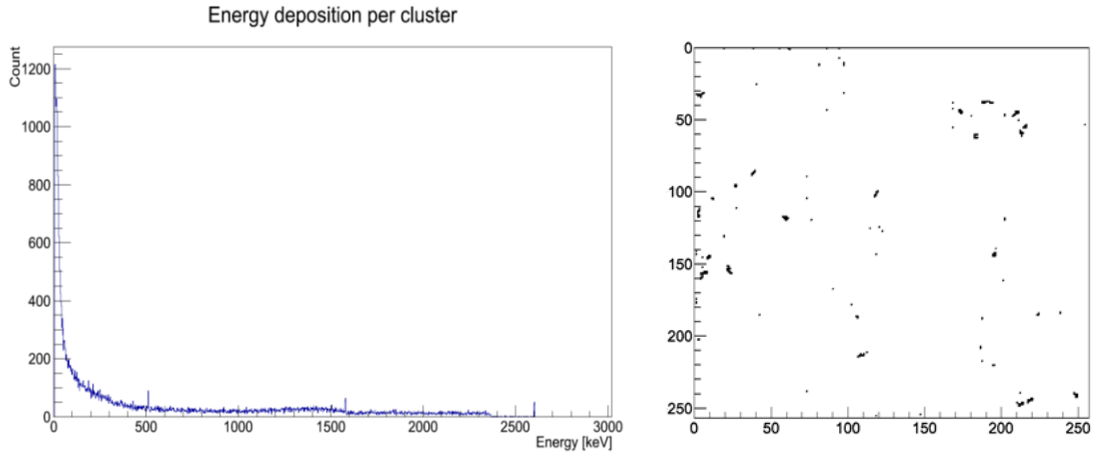


Figure 21: Simulated gamma particles characteristics (left: energy spectrum of gamma particles, right: shape of detected particles).

The cosmic muons were simulated as an important component of background radiation coming from the space, because muons detected by Timepix device are mostly from cosmic rays. A muon energy spectrum and examples of tracks can be found in Figure 22.

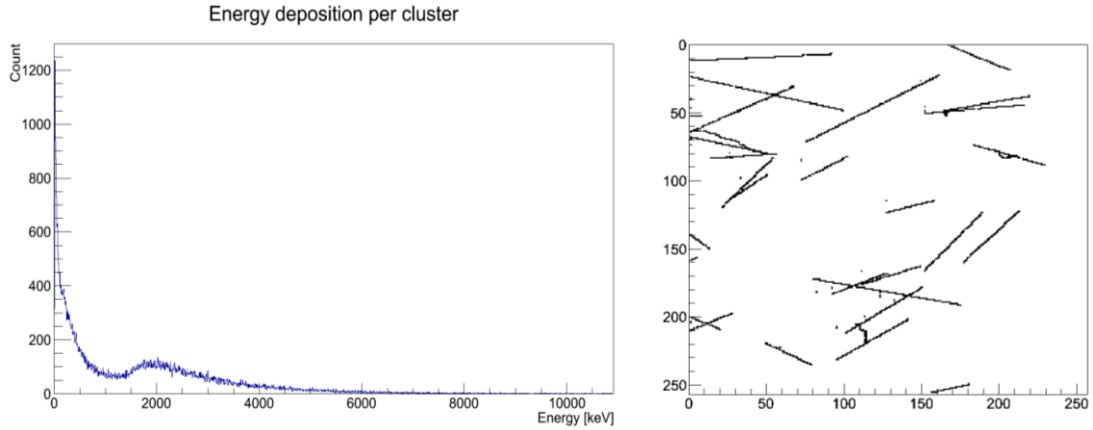


Figure 22: Simulated muons characteristics (left: energy spectrum of muons, right: shape of detected particles).

The proton particles data was measured at the VdG accelerator, therefore the data contains a small amount of residual gamma particles. However, the gamma was filtered out easily. Their energy spectrum and examples of tracks can be found in Figure 22.

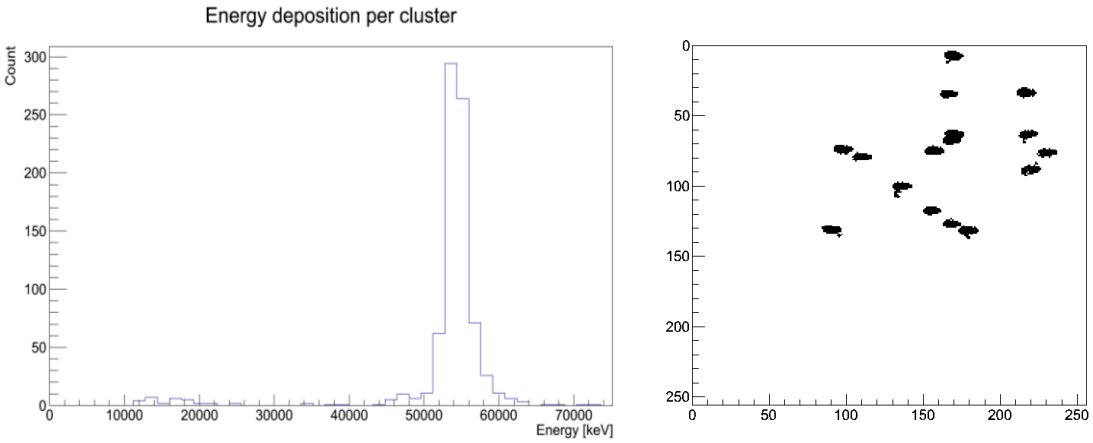


Figure 23: Protons characteristics (left: energy spectrum of protons, right: shape of detected particles).

4.1.2 Mixed data for additional testing

To check feasibility of used methods for classification, the additional experimental data were obtained. The data were detected by Timepix device using two sources ^{90}Sr-^{90}Y (electrons) and ^{241}Am (alpha particles). The Figure 24 illustrates characteristics of the data – in this case the difference in energy signatures for these two types of particles is obvious. Nevertheless, the deposited energy is not used as a classification criterion because it varies with the radioactive sources used for the measurement.

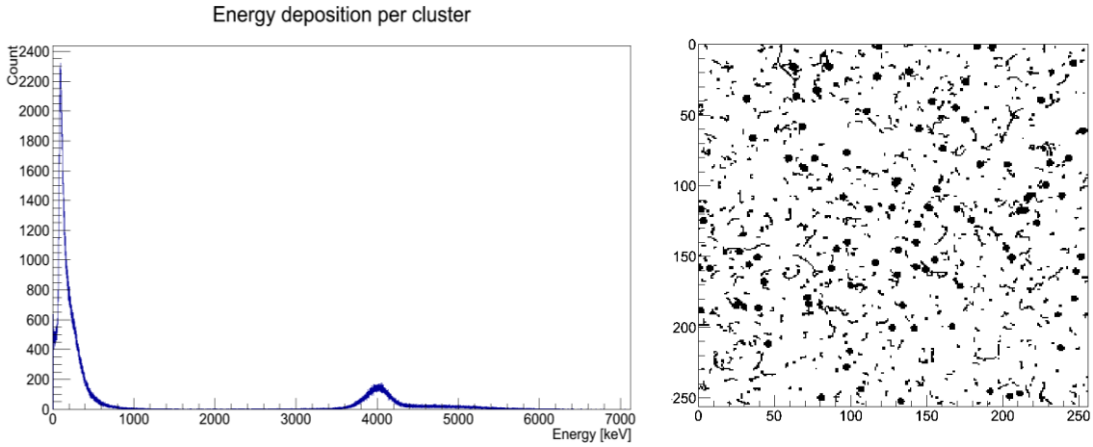


Figure 24: Experimental data characteristics (left: energy spectrum of experimental data, right: shape of detected particles).

4.2 Comparison of classification methods

This chapter presents a comparison of tested classification methods evaluated by the means of efficiency and background rejection. For each method, the graphs

were constructed for all simulated and measured data sets described in chapter 4.1.1. Three methods (MLP, BDT and LDA), which were described earlier in chapter 2.4, were used.

4.2.1 Multi-layer perceptron

The MLP implemented in the TMVA package offers many parameters than can be set. The parameters are described in the TMVA manual [21] on the page 94. We studied how the results depend on the settings of different parameters (see Table 1).

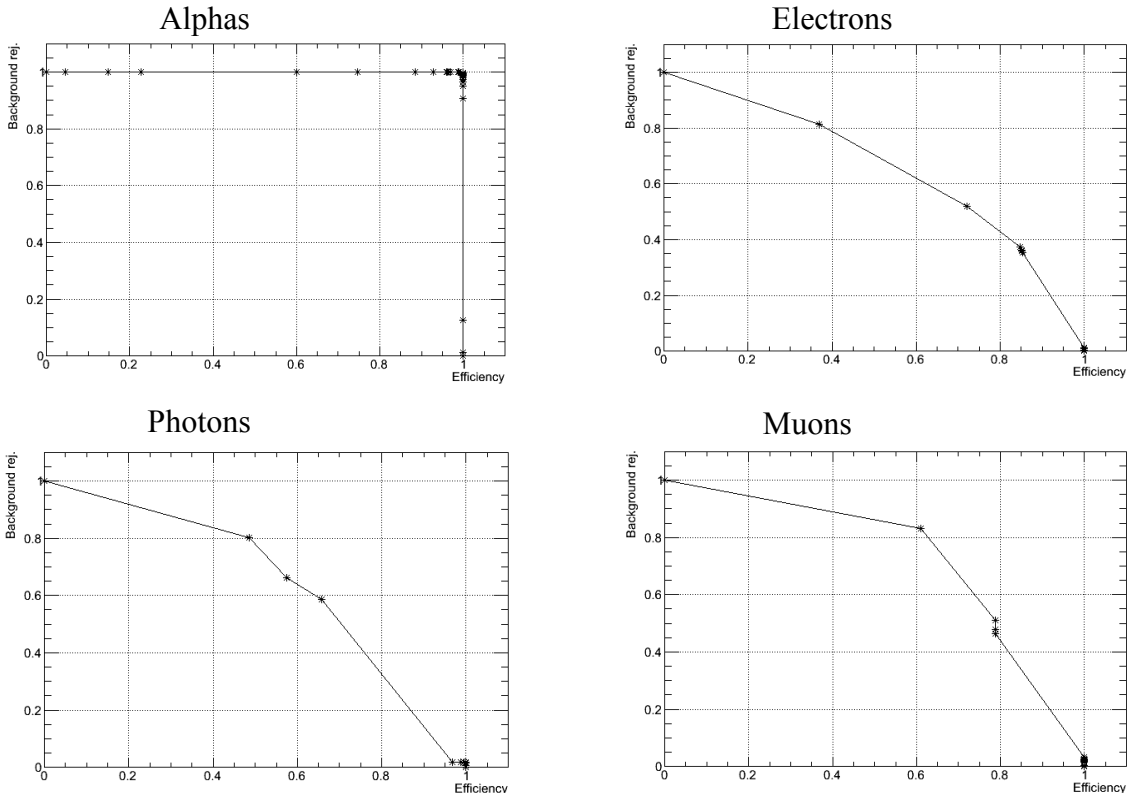


Figure 25: Background rejection vs. efficiency of selection with default MLP settings.

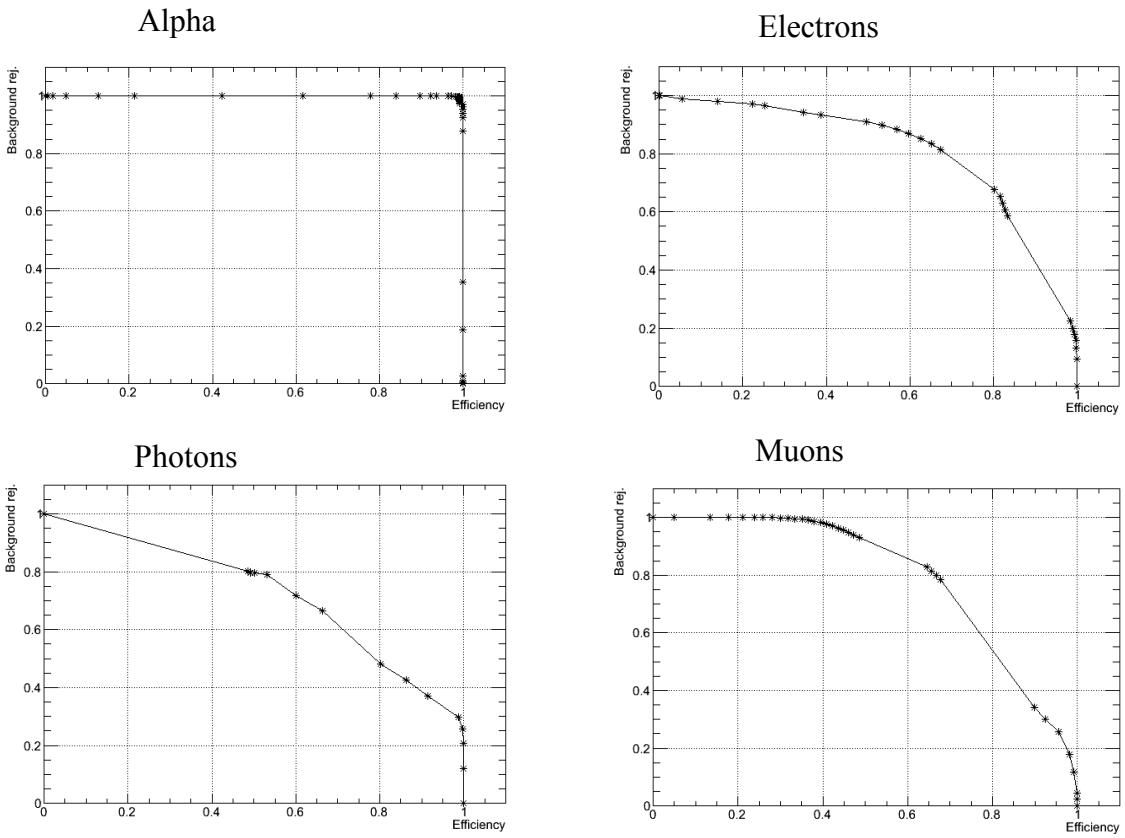


Figure 26: Background rejection vs. efficiency of selection with MLP with 3 hiddlen layers.

First of all, we tried to run the MLP procedure with default parameters (Figure 25). Then, we achieved better results using a MLP with three hidden layers (N+10, N+5, N neurons; Figure 26). This size proved to be optimal because adding more layers did not improve the result (Figure 27 with N+25, N+20, N+15, N+10, N+5, N neurons).

The TMVA package also offers more activation functions – linear, tanh and radial (the default function is sigmoid). The neural network with linear activation functions did not converge in the case of data sets (line 5 of Table 1). The radial and tanh activation functions gave approximately the same results as the sigmoid function. Changing the training method gave the same results as well. The default back propagation method proved to be the fastest one (this is also mentioned in [21]). The results might be found on the DVD attachment. We also tried changing other parameters but they did not seem to have any effect on the result except that the learning process did not converge for increased ANN (artificial neural network) learning rate parameters. The parameters of all significant experiments can be found in Table 1.

MLP parameters	Result
HiddenLayers=N+10, N+5, N	Success
HiddenLayers=N+25, N+20, N+15, N+10, N+5, N	Success
HiddenLayers=N+10, N+5, N;NeuronType=tanh	Success
HiddenLayers=N+10, N+5, N;NeuronType=radial	Success
HiddenLayers=N+10, N+5, N;NeuronType=linear	Fail
HiddenLayers=N+10, N+5, N:LearningRate=0.01:DecayRate=0.005	Success
HiddenLayers=N+10, N+5, N:LearningRate=0.021	Fail
HiddenLayers=N+10, N+5, N:LearningRate=0.025	Fail
HiddenLayers=N+10, N+5, N:LearningRate=0.025	Fail
HiddenLayers=N+10, N+5, N:LearningRate=0.0025:DecayRate=0.0012	Fail
TrainingMethod=BFGS	Success

Table 1: A list of MLP tests for different parameters. Selected parameters for tests are presented in the first column.

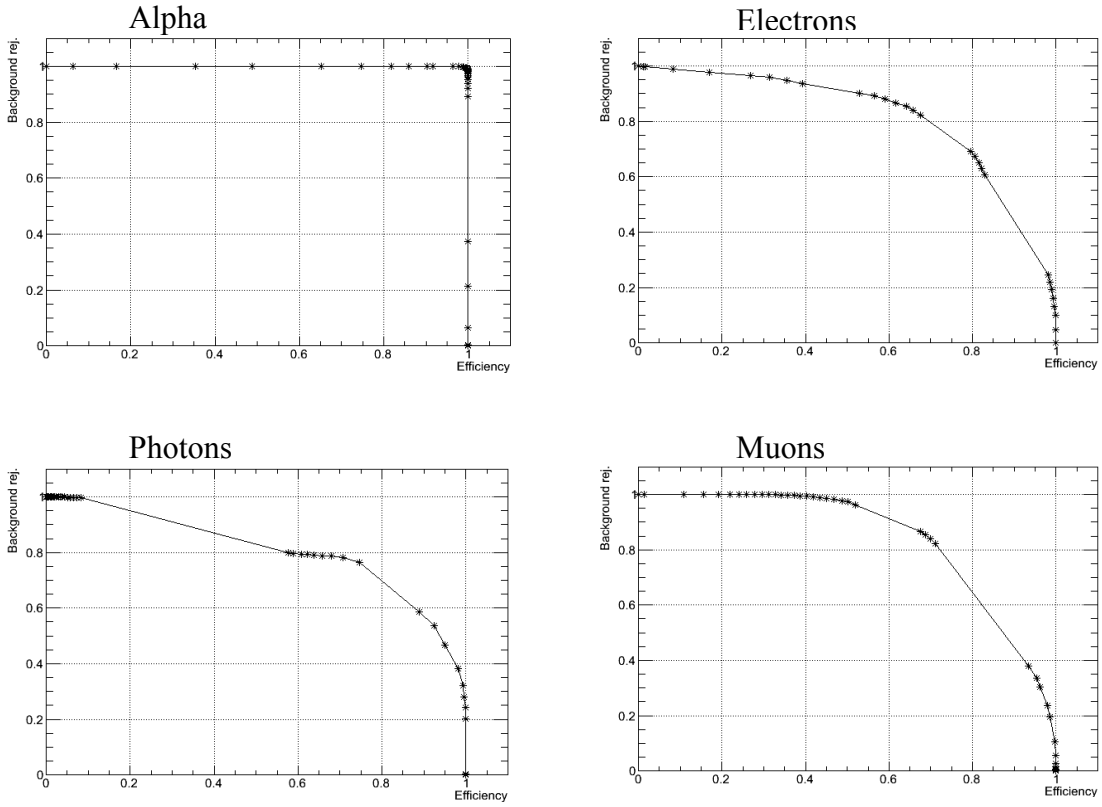


Figure 27: Background rejection vs. efficiency of selection with MLP with 5 hidden layers.

4.2.2 Boosted Decision Tree

The BDT method is less suitable for the track classification problems as it showed slightly worse results. This method is very stable and does not depend on many parameters. The only parameter it seems to depend is the boosting method. The software can use adaptive boosting method and gradient boosting method (see Figure 28 and Figure 29, respectively).

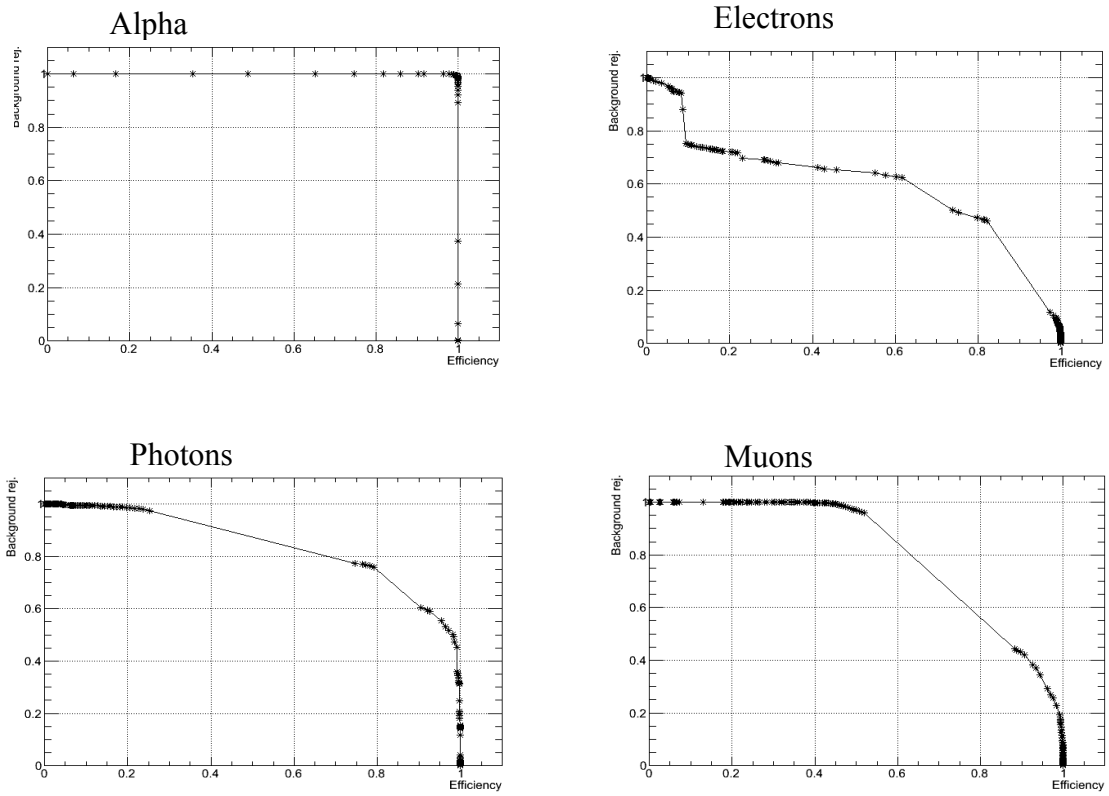


Figure 28: Background rejection vs. efficiency of selection with BDT with adaptive boost.

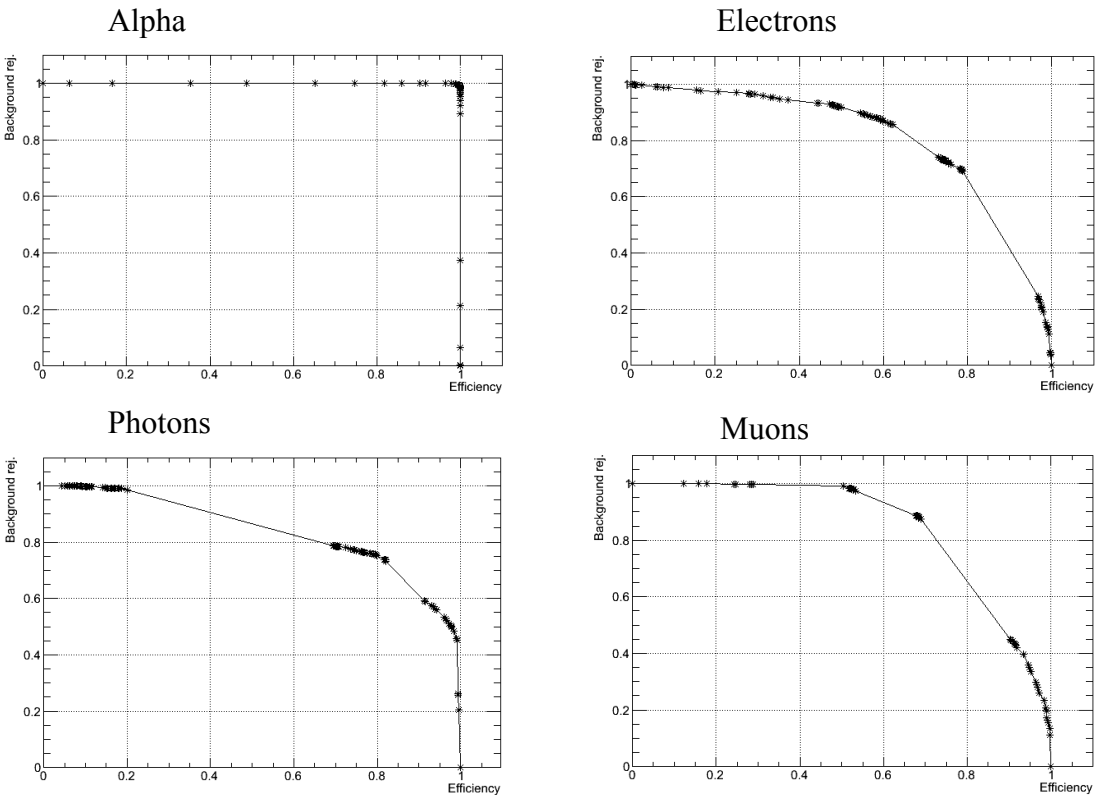


Figure 29: Background rejection vs. efficiency of selection with BDT with gradient boost.

4.2.3 Linear discriminant analysis

The LDA method gives the worst results compared to the other methods. See Figure 30.

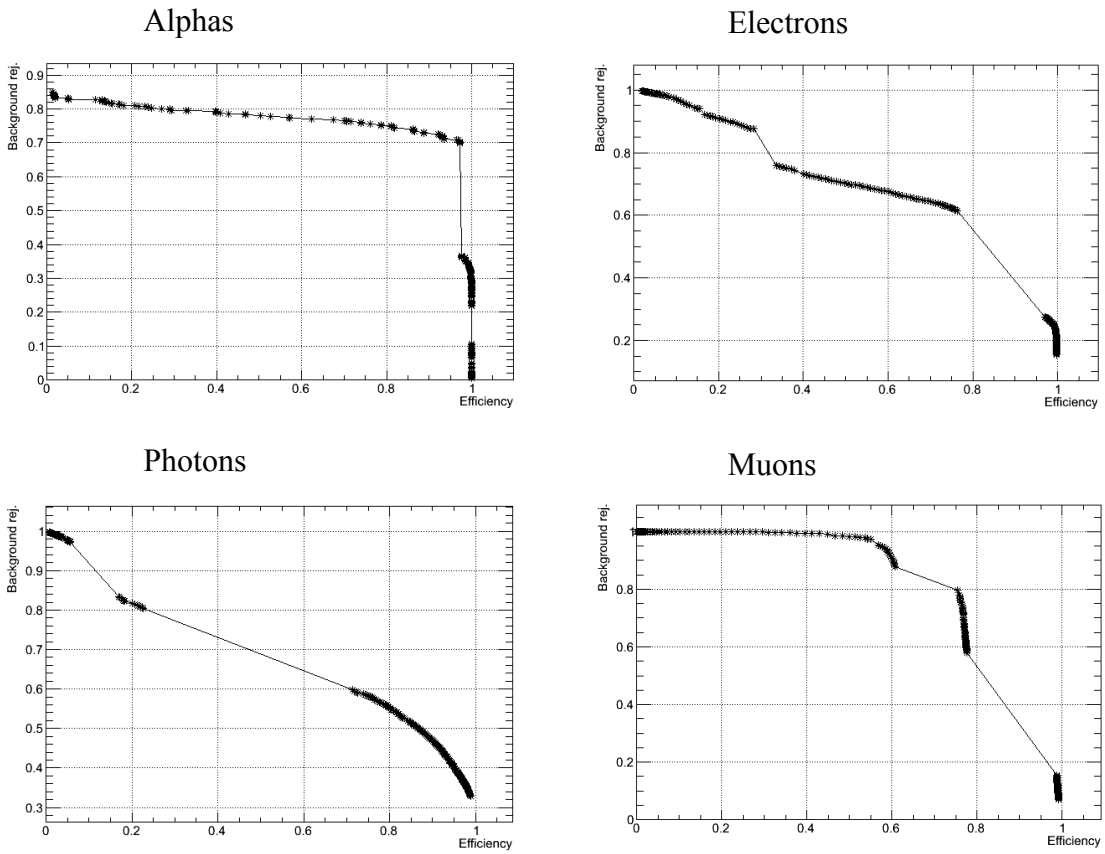


Figure 30: Background rejection vs. efficiency of selection with LDA.

4.2.4 Comparison of the methods

We evaluated three classification methods with several parameter settings with the data set described in chapter 4.1.1. The MLP method proved to be the most suitable method for this problem. The parameter settings are optimal using three hidden layers. Additional layers do not add any additional classification ability and it is slower due to its larger size. The BDT method is usually slightly worse than neural network but it is comparable. It outperforms the MLP in the case of muons. The optimal boosting method seems to be the gradient boosting method (described in chapter 2.4.2). The LDA classification ability seems to be the least suitable for our purposes.

The alpha particles are the easiest to recognize due to their distinctive shape. The second particles that are easy to classify are protons. However, their shape might vary depending on the measuring conditions, therefore this will be a subject of additional research. The muons are harder to recognize because they sometimes have very similar shape to the high energetic electrons. The gammas and electrons are the

most difficult to separate – the gamma particle often produces the same tracks as electron due to the Compton scattering. The selected efficiency (at the ends of the scale) and background rejection values are given in Table 2.

Particle type (used method)	Efficiency	Background rejection
Alpha (MLP)	97%	99.9%
Alpha (MLP)	100%	97%
Protons (MLP)	97%	99.9%
Protons (MLP)	99%	97%
Muons (MLP)	40%	98%
Muons (MLP)	98%	24%
Muons (BDT)	53%	98%
Muons (BDT)	98%	25%
Electron (MLP)	15%	98%
Electron (MLP)	98%	25%
Gamma (MLP)	5%	98%
Gamma (MLP)	98%	30%

Table 2: Selected efficiency of the selection and background rejection values.

4.2.5 Proton data

We learned the most suitable method (MLP with 3 hidden layers) with data set extended with the proton data. Results are presented in Figure 31 and Table 2.

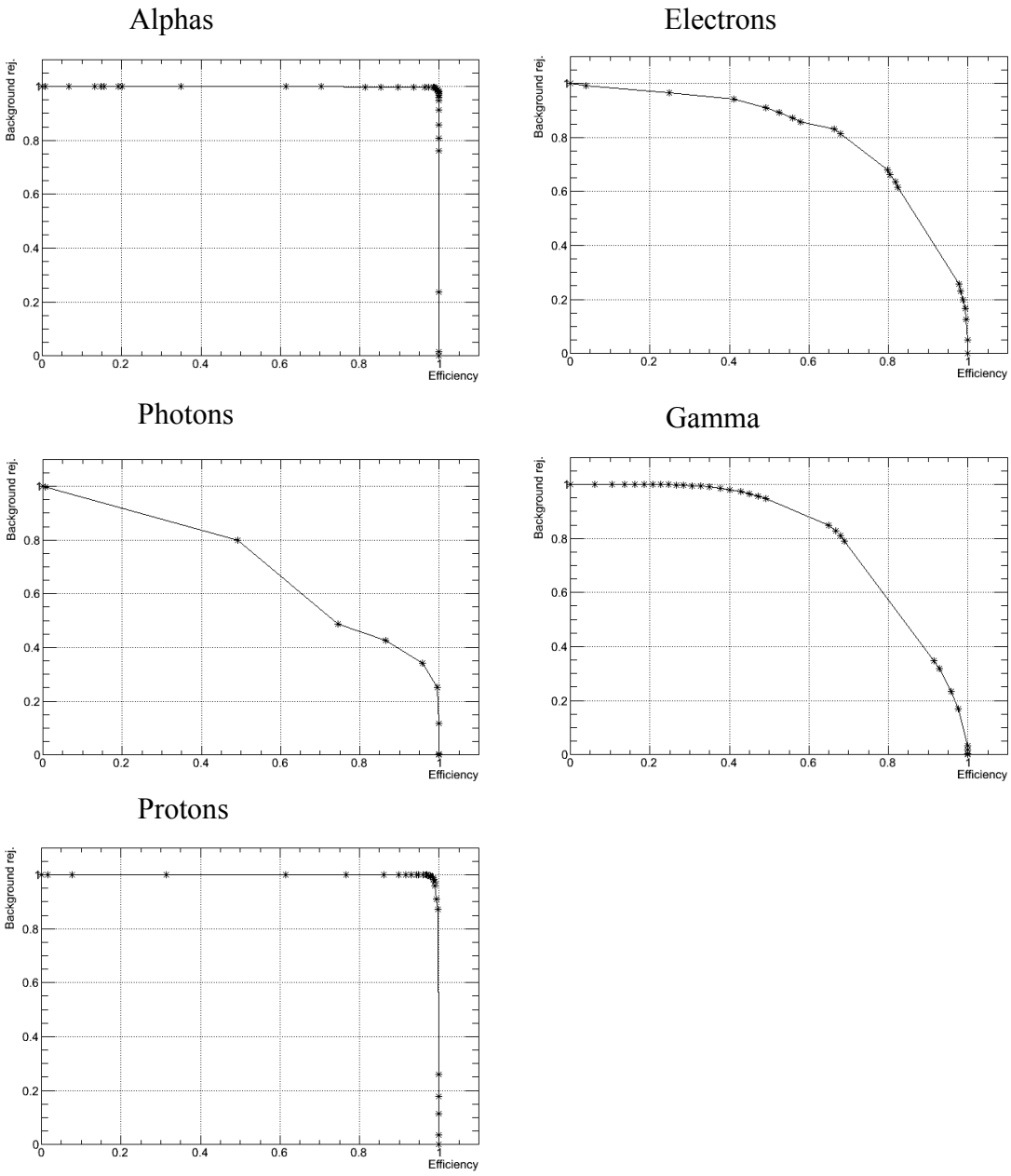


Figure 31: Background rejection vs. efficiency of selection with MLP with 3 hidden layers.

4.3 Experimental data

Additional test of the Pixa software was performed on a set of experimental data (the data set described in chapter 4.1.2), which is a mixture of electrons and alpha particles. This data set was measured with different type of the detector and has different energy spectrum than the simulated data (described in chapter 4.1.1). The results are given in Table 3. The clusters smaller than 4 pixels do not participate in

the classification because they are too small to have any distinctive shape. In real experiment, they correspond to low energy electrons.

(1)	Number of clusters in the dataset:	2,062,398
(2)	Number of clusters from ^{241}Am source (alphas):	404,798
(3)	Number of clusters (1) classified as alphas:	380,208
(4)	Number of alpha clusters (2) classified as alphas (correct signal):	379,044
(5)	Number of electron clusters (6) classified as alphas (background classified as signal):	1164
	Efficiency for alphas (4) / (2)**:**	**93.6%**
	Background rejection for alphas 1 – (5) / (6)**:**	**99.9%**
(6)	Number of clusters from ^{90}Sr source (electrons):	1,657,600
(7)	Number of clusters (1) classified as electrons:	1,282,822
(8)	Number of electron clusters (6) classified as electrons (correct signal):	1,282,735
(9)	Number of alpha clusters (2) classified as electrons (background classified as signal):	87
	Efficiency for electrons (8) / (6):	**77.4%**
	Background rejection for electrons 1 – (9) / (2)**:**	**99.98%**
	Selected MVA cuts	0.3 (electron), 0.35 (alpha)

Table 3: Experimental data evaluation results.

The results can be also illustrated in the cluster size distribution histogram (Figure 32 with whole dataset and Figure 33 with separated data). It shows that the two intersecting curves are clearly separated from each other. The MVA cuts were selected considering we wanted 90% background rejection estimated on the charts given in Figure 26.

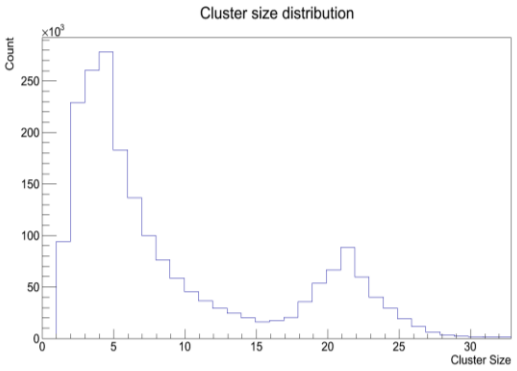


Figure 32: Cluster size distribution of the dataset.

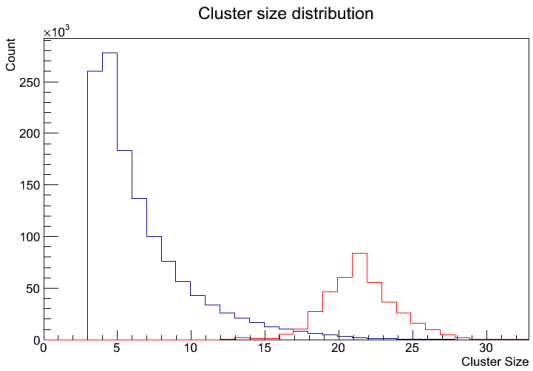


Figure 33: Cluster size distribution of selected electrons (blue) and alphas (red).

5 Conclusion

In this thesis, we focused on a classification of particle tracks in the Timepix pixel detector. We presented criteria that can be used to describe a shape of the particle track, listed some of the most popular methods used in the data-mining field for a classification and evaluated the method giving the best results. The software called Pixa was implemented. This software can be used for a processing of a broad range of data originating from a Timepix detector. The main goal of this thesis was to create software package offering quantitative particle track recognition (with exact defined efficiency of selection and background rejection).

This software not only uses the above mentioned classification methods, but implements also many filters and routines to process the experimental data and gives meaningful physical results. During the processing, it can create pixel masks according to various criteria and apply them to the data. It provides energy calibration procedures, transformation of data between different formats (corresponding to different historical versions of the acquisition software) and definition and application of a complex energy threshold to the data.

The methods and software implemented in this thesis was already used for experimental data processing and the results were published in several papers ([4], [30], [31] and [32]; listed as a co-author). The developed software Pixa is extensively used in two running experiments of double beta decay experiment TGV (2νEC/EC decay of ^{106}Cd) and COBRA ($\beta^-\beta^-$ of several isotopes, such as ^{116}Cd).

Although it is not a modular software, it can be modified easily; the new tasks and methods are registered in a dynamic way to remove unwanted dependencies between the layers of the code. The data processing layer (the processing routines) can be also used as a library with a little additional work.

The portability of Pixa software was considered less important during the development time. The processing core of the software is more or less portable but the GUI is bound to the MFC library, which works only on Windows operating systems. Nevertheless, a portable version of the Pixa based on the Qt is under development. This would allow us to run Pixa on Mac OS and Linux.

In future work we plan a further testing of Pixa software capabilities on very complex experimental data (high density, overlap of particles of different types).

More attention will be also paid to more advanced learning and classification process taking into account also a distortion of cluster between two neighbouring pixel detectors due to their position at the edge of the detector, or discontinuity of cluster due to the non-zero threshold of each pixel.

6 Bibliography

[1] X. Llopart et al., "Timepix, a 65k programmable pixel readout chip for arrival time, energy and/or photon counting measurements," *Nucl. Inst. Meth.,* vol. 581, no. 1-2, pp. 485-494, 2007.

[2] J. Jakůbek, "Semiconductor Pixel detectors and their applications in life sciences," *Journal of Instrumentation,* vol. 4, P03013, 2009.

[3] "http://www.cobra-experiment.org/," [Online].

[4] V. B. Brudanin, V.G. Egorov, A. A. Klimenko et al., "Summary of the TGV experiment and future plans," *AIP Conference Proceedings,* vol. 1417, pp. 110-114, 2011.

[5] A.S. Barabash, "Double beta decay experiments," *arXiv:1107.5663v1 [nucl-ex]*.

[6] C. Arnaboldi et al. , "CUORE: A cryogenic underground observatory for rare events," *Physics reports-review section of physics letters,* vol. 307, no. 1-4, pp. 309-317 , 1998.

[7] I. Abt et al., "A New 67Ge Double Beta Decay Experiment at LNGS," *arXiv:hep-ex/0404039,* 2004.

[8] Danilov M et al., "Detection of very small neutrino masses in double-beta decay using laser tagging," *Phys. Lett.,* vol. B 480, p. 12, 2000.

[9] R. Saakyan et al., "Topological detection of double beta decay with NEMO3 and SuperNEMO," *J. Phys. Conf. Ser.,* vol. 179, 012006, 2009.

[10] M. Platkevič et al., "Signal processor controlled USB2.0 interface for Medipix2 detector," *Nucl. Instr. And Meth. in Phys. Res. A,* vol. 591, no. 1, pp. 245-247, 2008.

[11] "Pixelman Package," IEAP, [Online]. Available: http://aladdin.utef.cvut.cz/ofat/others/Pixelman/Pixelman.html. [Accessed 11 March 2013].

[12] J. Bouchami, et al., "Study of the charge sharing in silicon pixel detector by means of heavy ionizing particles interacting with a Medipix2 device," *IEEE Nuclear Science Symposium Conference Record,* Vols. 1-9, pp. 633-635, 2009.

[13] V. Morard et al., "Efficient geodesic attribute thinnings based on the barycentric diameter," *J. Mathematical Imaging and Vision,* vol. 46, no. 1, pp. 128-142, 2013.

[14] C. Lantuejoul, S. Beucher, "On the use of the geodesic metric in image analysis," *Journal of Microscopy,* vol. 121, pp. 39-49, 1981.

[15] J. Bartovsky et al., "Morphological classification of particles recorded by the Timepix detector," *Image and Signal Processing and Analysis (ISPA), 2011 7th International Symposium on,* pp. 343-348 , 2011.

[16] C. Lantuejoul, F. Maisonneuve, "Geodesic methods in quantitative image analysis," *Pattern recognition,* vol. 17, no. 2, pp. 177-187, 1984.

[17] J. Hershberger and S. Suri, "Matrix searching with the shortest path metric," *SIAM J. Comput.,* pp. 485-494, 1993.

[18] S. W. Bae, M. Korman and Y. Okamoto, "The Geodesic Diameter of Polygonal Domains," *Lecture Notes in Computer Science,* vol. 6346, pp. 500-511, 2010.

[19] R. A. Fisher, The Use of Multiple Measurements in Taxonomic Problems, 1936.

[20] T. Hastie et al, The Elements of Statistical Learning, Springer, ISBN 978-0387952840, 2013.

[21] A. Hoecker, P. Speckmayer, J. Stelzer, J. Therhaag, E. v. Toerne and H. Voss, "TMVA - Toolkit for Multivariate Data Analysis" *PoS ACAT 040 (2007), arXiv:physics/0703039.*

[22] P. D. Wasserman, T. Schwartz, "Neural networks. II. What are they and why is everybody so interested in them now?," *IEEE Expert,* vol. 3, no. 1, pp. 10-15, 1988.

[23] S. Haykin, Neural Networks: A Comprehensive Foundation (3rd ed), Prentice Hall, ISBN 978-0131471399, 2008.

[24] S. Theodoridis and K. Koutroumbas, Pattern Recognition, Academic Press, ISBN 9781597492720, 2009.

[25] M. Aizerman et al, "Theoretical foundations of the potential function method in pattern recognition learning," *Automation and Remote Control ,* p. 821–837, 1964.

[26] D. Tureček et al., "Energy calibration of pixel detector working in Time-Over-

Threshold mode using test pulses," *IEEE Nuclear Science Symposium Conference Record,* pp. 1722-1725, 2011.

[27] "http://www.codeproject.com/Articles/212856/Head-to-head-benchmark-Csharp-vs-NET," [Online].

[28] R. Brun and F. Rademakers, "ROOT - An Object Oriented Data Analysis Framework," in *Proceedings AIHENP'96 Workshop, Lausanne, Sep. 1996, Nucl. Inst. & Meth. in Phys. Res. A 389*, 1997.

[29] Microsoft, "MSDN Library," [Online]. Available: http://msdn.microsoft.com/en-us/library/4x1xy43a%28VS.80%29.aspx.

[30] P. Čermák et al., "Use of silicon pixel detectors in double electron capture experiments," *Journal of instrumentation,* vol. 6, C01057, 2011.

[31] J. M. Jose et al., "Timepix background studies for double beta decay experiments," *Journal of instrumentation,* vol. 6, C11030, 2011.

[32] N. I. Rukhadze et al., "Ispolzovanie pixelnych detektorov isledovanijach EC/EC raspada," *Izvestija ran, Seria fizićeskaja,* vol. 77 (4), pp. 420-423, 2013.

[33] "Wikipedia," [Online]. Available: http://en.wikipedia.org/wiki/Artificial_neural_network. [Accessed 3 3 2013].

7 Publications

7.1 Publication related to this thesis

[4] V. B. Brudanin, V.G. Egorov, A. A. Klimenko et al., "Summary of the TGV experiment and future plans," *AIP Conference Proceedings,* vol. 1417, pp. 110-114, 2011.

[30] P. Čermák et al., "Use of silicon pixel detectors in double electron capture experiments," *Journal of instrumentation,* vol. 6, C01057, 2011.

[31] J. M. Jose et al., "Timepix background studies for double beta decay experiments," *Journal of instrumentation,* vol. 6, C11030, 2011.

[32] N. I. Rukhadze et al., "Ispolzovanie pixelnych detektorov isledovanijach EC/EC raspada," *Izvestija ran, Seria fizićeskaja,* vol. 77 (4), pp. 420-423, 2013.

7.2 Other publications

F. Blashke et al., “CZELTA: An overview of the Czech large-area time coincidence array,” *Astrophysics and Space Sciences Transactions*, 7 (1), pp. 69-73, 2011.

K. Smolek, et al., “Measurement of high energy cosmic rays in the experiment CZELTA,” *IEEE Nuclear Science Symposium Conference Report*, pp. 720-721, 2008.

8 List of Figures

9 List of Tables

Appendix A - DVD-ROM

The enclosed DVD contains:

- source code of the Pixa software,
- executable and other files required to run Pixa software,
- installation files of required frameworks (Visual C++ Redistributable and ROOT framework),
- sample files with Pixa processing settings,
- sample results of Pixa software.

Appendix B - User documentation

B.1 System requirements

The Pixa software requires PC with Windows XP or higher. It requires to have the ROOT framework[3] and Visual C++ 2010 SP1 runtime installed.

B.2 Installation

The Pixa software requires having ROOT framework[3] installed first. Then, it is possible to install the application just by copying the files to any folder you want.

B.3 Software workflow

The software reads the measured and other data (defined in form of "data storage") and processes them using the tasks that are ordered in a chain. Each task does either an analysis (it creates histograms or other kind of (statistical) result) or filtering (it filters the data). Each task's output is the next task input – e.g. if some task filters out all cluster smaller than 5 pixels, all tasks that come after it processes only clusters larger or equal than 5 pixels. The tasks write their output into a common combined ROOT file or they can create their own output files (for special kind of tasks that usually creates some data based on the measured input data). A ROOT file is a special file format where we can store histograms, graphs and other objects or data in a hierarchical structure[4].

B.4 Main application window

In the Pixa, there is a single main window that contains all controls required to perform the processing. The main window consists of 8 main parts:

1. Menu and toolbar
2. List of data storages
3. Runtime statistics
4. List of processing chain tasks
5. List of task local filters
6. Properties of the currently selected object

[3] Available at http://root.cern.ch/drupal/content/downloading-root

[4] A better explanation can be found on http://root.cern.ch/drupal/content/root-files-1

7. List of files produced by tasks
8. Processing run controls.

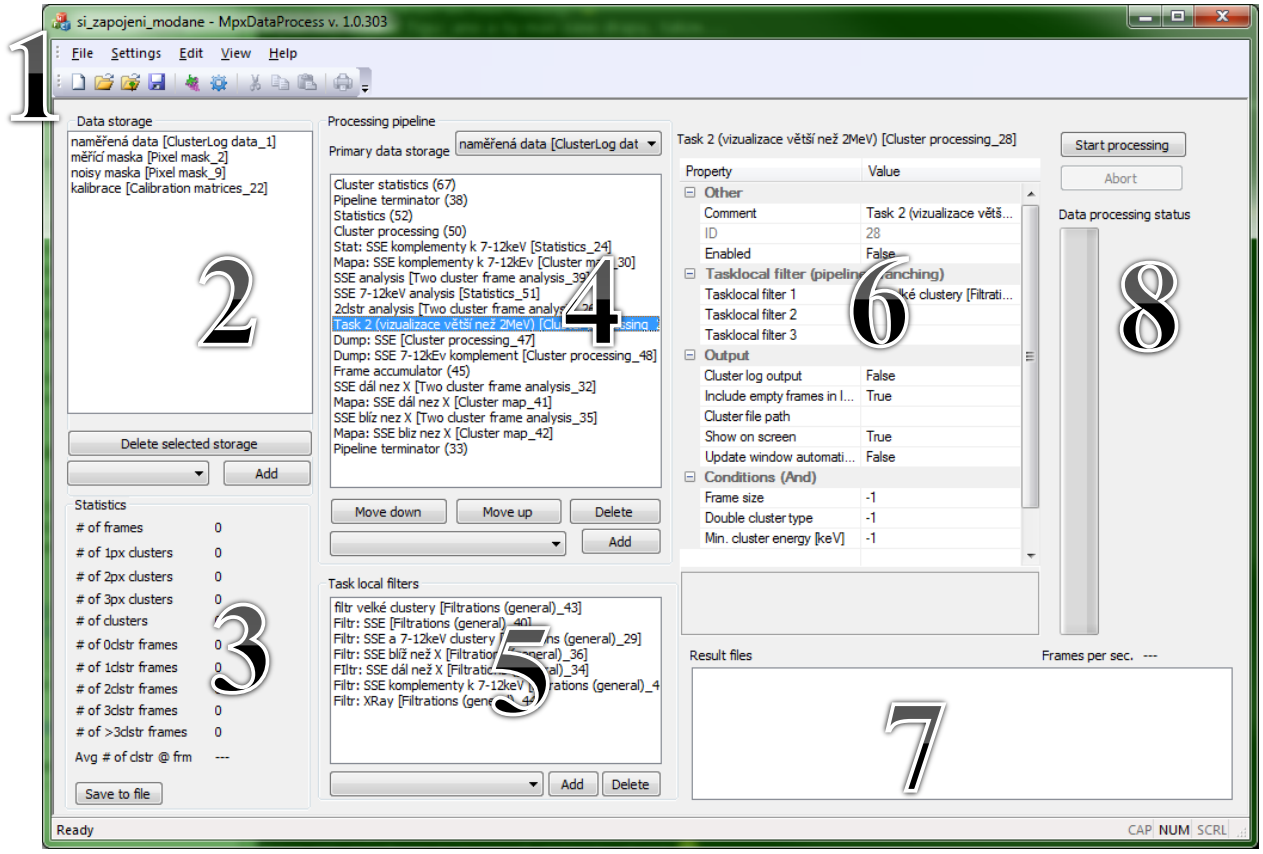


Figure 34: A main window of Pixa software tool

The File menu (1) allows user to load or save processing settings from a file. It can also load processing settings from the output file (which is created during the processing run) – it loads the same settings that were used to perform the processing. The Cluster settings allows you to set several parameters regarding how tasks work with clusters. Also, you can set the parameters (e.g. a path to a root file containing the learned data) and method for cluster classification.

The list of data storages (2) allows the user to set location and other parameters of all input files. The user should set at least some measured data. It is possible to include pixelmask for filtration and calibration data, those data have to be selected in the measurement data properties.

The runtime statistics (3) show various informational values about the data that are currently processed. The data are updated in real time, as they are read.

The list of processing chain tasks (4) contains all tasks in the processing chain. The user can add, delete tasks or change their order. The tasks are executed in given order and output of each task is used as an input of the next task (for more information and examples, see chapter 3.4). The parameters of the task can be set by clicking on the task and setting the parameters in the properties control (6). The primary data storage is a data storage that read measured frames that are processed by the tasks in the chain.

The list of task local filters (5) contains definition of all filtration tasks that can be used as a task local filter. The task local filter is a special kind of task that filters input data only for the task, where it is selected. Therefore, any other task is not affected by the task local filter. They are set in the properties control. They must me defined before they can be used. For more information and examples, see chapter 3.4.

The properties control (6) allows user to set properties of the selected object. The object can be a data storage, a task or settings group (output file settings or cluster settings).

The tasks can have one or more result files. In addition, there is always a single "output.root" file, which contains combined output of the tasks and a text file with simple overview statistics of the measured data. Those files are written in List of files produced by tasks (7). A double click on the file opens the file in a default application (which is set in the operating system).

The "start processing" button (8) starts the processing; the abort button aborts the processing. The progress bar show what portion of data has been read so far.

B.5 Frame viewer dialog

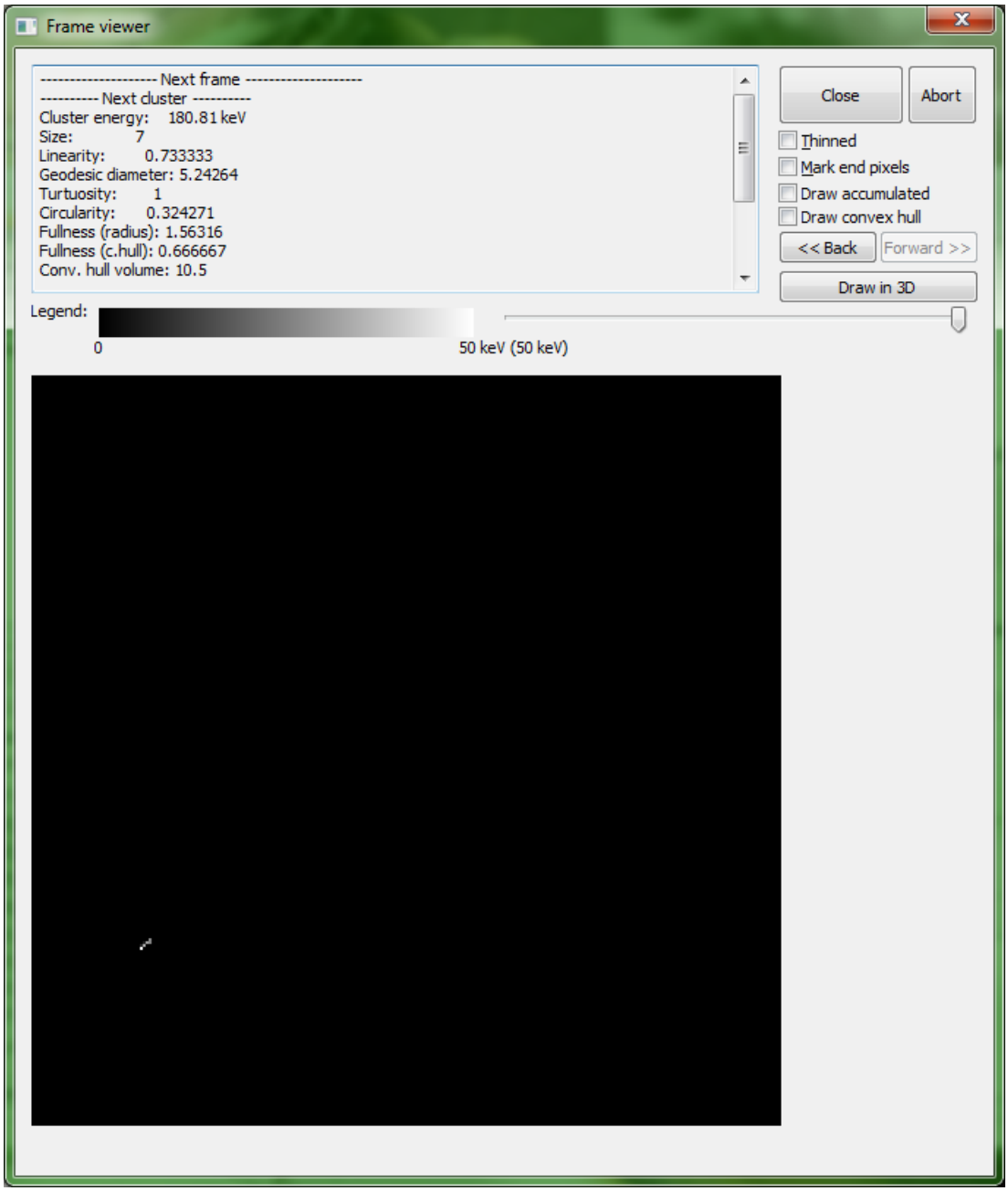


Figure 35: The frame viewer dialog

The frame viewer dialog (Figure 35) is used for viewing the frames and for showing the values of cluster properties of all clusters in the frame. The big black field contains the image of the frame – the black color corresponds to the empty pixels. The brighter the pixel, the more deposited energy it contains. The scale is shown above the frame image. The user can change the scale with the slider at the right side.

It is possible to show the previous or next frames (with the “Back” and “Forward” buttons) and to view several properties of the cluster in both text form (the big text box) and visualized form (the checkboxes).

B.6 Output.root contents

Each processing run creates a single output.root file. This file is placed at the location specified at the Output settings. The file contains data created by all tasks in the chain that creates any data (excluding a few tasks that puts their output into a separate file – for example noisymask creation or the learning task). The result data are structured into ROOT folders – each task (each instance) has a single folder, which contains all the histograms, graphs etc. These can be structured into other folders, creating a hierarchical structure (like in Figure 36).

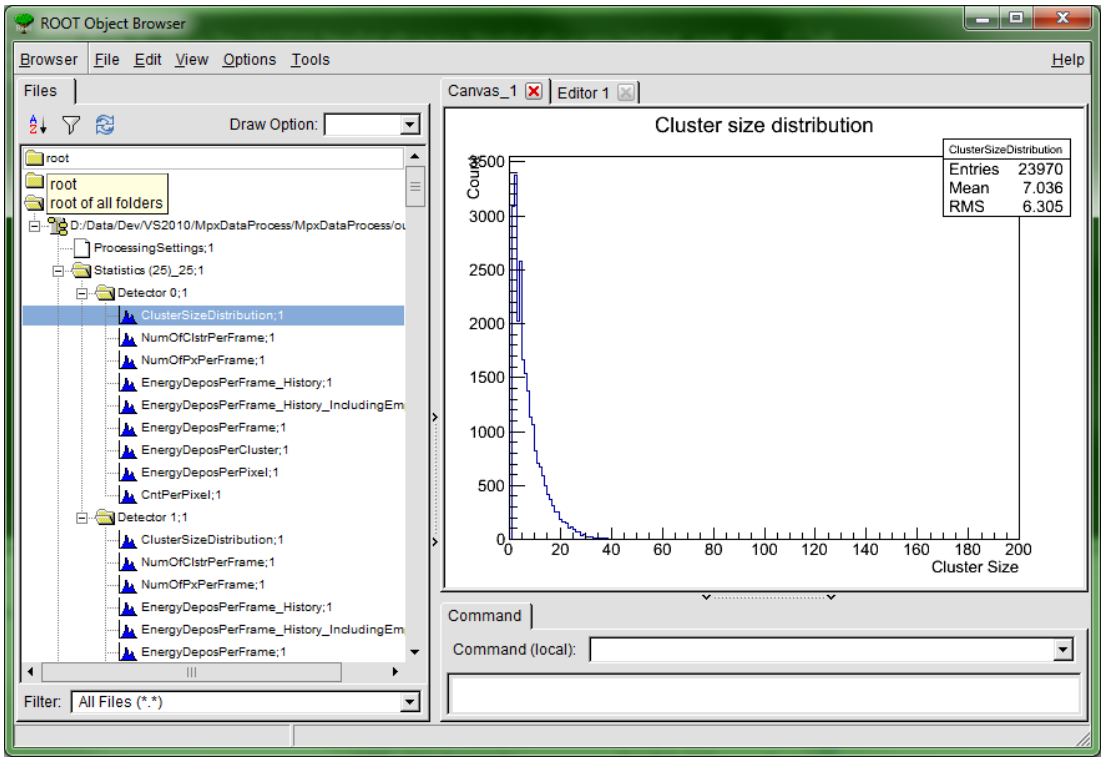


Figure 36: An example of hierarchical output.root structure

B.6.1 ROOT object browser

The ROOT object browser is a tool included in the ROOT package. It is used to view contents of the ROOT files. It has an Explorer-like interface: the left pane consists of hierarchical list of folders and ROOT objects and the right pane contains visualization of the selected object. The object can by selected by a double click.

This browser is registered automatically as a default program for the .root files by the ROOT installation.